Model Transformation of Model Fragments Using Borrowed Context: Extended Version

Hanna Schölzel

Bericht-Nr.
ISSN

Model Transformation of Model Fragments Using Borrowed Context: Extended Version

Hanna Schölzel

hannas@cs.tu-berlin.de, Institut für Softwaretechnik und Theoretische Informatik, Technische Universität Berlin, Germany

Abstract

In this technical report we study the transformation of models in the context of algebraic graph transformation and triple graph grammars. The new contribution of the thesis is to define and analyze the transformation of model fragments in general and the propagtion of graph constraints in particular.

With the borrowed context we developed a technique further to the model transformation with triple graph grammars. This allows a transformation of incomplete models which could not be transformed until now.

Moreover, we defined under which conditions a graph constraint can be propagated with borrowed context transformations and the model properties are preserved. This is also analyzed in the case study using the modeling framework ABT-Reo.

1 Introduction

In Computer Science models and visual languages become more and more important. They are most commonly used in the progress of specification, as well as for maintaining existing systems. Moreover, models and visual languages play an important role in other fields of work as well, like construction plans or even business models, which are used in the first case study of this thesis.

In contrast to a written language, the information provided through a visual language can be gathered at a few glances. It is fast and easy to understand, because the observer can make fast associations while viewing the picture.

But instead of mere pictures visual languages and their interpretation are defined. This can be either descriptive or constructive, similiar to a spoken language. In this diploma thesis the constructive approach of describing visual languages with algebraic graph grammars will be used based on the Double Pushout Approach introduced by [Ehr79] to transform graphs.

Furthermore, it focuses on model transformation with Triple Graph Grammars [Sch94]. This technique is commonly used for the algebraic transformation of complete, correct models between model languages using Triple Graph Grammars. Since 1994, extensions of the original TGG definitions have been published [KS06] and applications have been presented [KW07]. However, one may stumble upon problems, if the model to transform is incomplete, i.e., a model fragment, since the triple graph grammars with their rules are designed for complete models and they are not necessarily applicable to model fragments.

In order to solve this problem the approach of the Borrowed Context approach [EK06] is used in this thesis. Originally, this approach was developed for the purpose of deriving bisimulation congruences from unlabelled reaction rules. It will be adjusted to the context of model transformation in order to make graph transformation rules applicable in a smaller context. Therefore, missing parts for the transformation are borrowed from the left-hand-side of the transformation rule and added to the model fragment, resulting in a larger model where the rule is applicable.

In practice, model fragments appear as parts of graph constraints, for instance. These are Boolean formulas, which define specific conditions a model has to satisfy, i.e., it requires or forbids certain structures in the model. Such structures are not necessarily complete models, but parts of them. A main result of this thesis is the propagation of graph constraints. With the use of the borrowed context approach a graph constraint can be translated completely and it will be shown, under which conditions a model from the target language satisfies the propagated constraint.

In a case study this theory will be applied, to transform a graph constraint in the modeling framework ABT-Reo [BHE09], [BH09], which was invented for the organization Credit Suisse for enterprise modeling. It combines IT and business models in a triple graph grammar and we will show an example propagation of an IT constraint to the business domain. Additionally, we will show, that the needed conditions are fulfilled, such that the propagated model is an integrated constraint for both - IT and business - domains.

The thesis is structured as follows. In Section *2 Review of Model Transformation*

with Triple Graph Grammars the basics of the algebraic graph transformation introduced in [EEPT06] will be presented, which are essential for the definition of triple graphs. The notion of triple graphs and triple graph transformation is the basis for the model transformation technique used in this work and will also be presented in this Section.

The theory of the borrowed context will be introduced in Section *3 Model Transformation with Borrowed Context*. Furthermore we will present conditions under which a transformation with borrowed context can be extended to a standard double pushout transformation. Additionaly we will show under which conditions a forward transformation with borrowed context is source consistent and introduce an on-the-fly construction for model transformation sequences with borrowed context. In the last part of this Section we will present the notion of correctness, completeness and termination for borrowed context based transformations.

In Section *4 Propagation of Graph Constraints Using Borrowed Context* we will introduce our main result. We define how to propagate a graph constraint with the previously introduced model transformation with borrowed context. Additionally we will show under which conditions an integrated model satisfies the integrated constraint.

Section *5 Case Study* presents in the first three subsections the propagation of a graph constraint with the borrowed context approach in the modeling framework ABT-Reo. We will give a short overview on the triple graph grammar ABT-Reo and show an example transformation in the first part of the Section. Afterwards, the technique developed in the previous section will be used to propagate a graph constraint. Moreover, we will show, that the triple graph grammar satisfies the necessary conditions for the propagation by analyzing it with the software tool AGG [AGG] for attributed graph grammar systems.

In the last part of this Section we present a case study on the tranformation of operational semantics with borrowed context transformations between State Machines [EEPT06] and Petri Nets [Rei84]. We study the possibility to transform operational semantics and preserving the model properties, i.e., the equivalence of first applying rules of the operational semantic to a State Machine and then transforming it into a Petri Net and vice versa.

In Section *6 Conclusion* we give an overview of the achieved results and of future work based on the results of this thesis.

2 Review of Model Transformation with Triple Graph Grammars

This section will be an introduction to the basics of model transformation. In section *2.1 Graph Transformation* the general concept of graphs and the algebraic graph transformaion based on the double-pushout approach will be introduced. An introduction to the mechanics of model transformation with triple graphs grammar will be given in section *2.2 Triple Graphs*.

2.1 Graph Transformation

The model transformation approach used in this thesis is algebraic graph transformation. Because this is a formal approach, it is well suited for formal analyses. On the other hand, due to the fact that graphs are visual and schematic, they are also a way of explaining complex situations on an intuitive level. In this section we define the basic notion of graphs and graph morphisms including typed graphs, which are also the basis for typed attributed graphs (see [EEPT06]) used in our case study.

Definition 1 (Graph).
A graph $G = (V, E, s, t)$ consists of a set V of nodes (also called vertices), a set E of edges, and two functions $s, t : E \to V$, the source and target functions:

$$E \underset{t}{\overset{s}{\rightrightarrows}} V$$

Graphs can be related by mapping the nodes and edges of a graph to those of another one under the condition, that the source and the target of each edge are preserved. The formal construct for this is called graph morphism.

Definition 2 (Graph Morphism).
Given graphs G_1, G_2 with $G_i = (V_i, E_i, s_i, t_i)$ for $i = 1, 2$, a *graph morphism* $f : G_1 \to G_2$, $f = (f_V, f_E)$ consists of two functions $f_V : V_1 \to V_2$ and $f_E : E_1 \to E_2$ that preserve the source and target functions, i.e., $f_V \circ s_1 = s_2 \circ f_E$ and $f_V \circ t_1 = t_2 \circ f_E$

$$\begin{array}{ccc} E_1 & \underset{t_1}{\overset{s_1}{\rightrightarrows}} & V_1 \\ {\scriptstyle f_T}\downarrow & = & \downarrow{\scriptstyle f_V} \\ E_2 & \underset{t_2}{\overset{s_2}{\rightrightarrows}} & V_2 \end{array}$$

A graph morphism f is *injective* (or *surjective*) if both functions f_V, f_E are injective (or surjective, respectively); f is called *isomorphic* if it is bijective, which means both injective and surjective.

Graphs and morphisms defined above lead to the category **Graphs**. In general, a category is a mathematical structure that has objects and morphisms, with a composition operation on the morphisms and an identity morphism for each object.

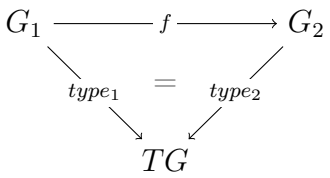
Definition 3 (The Category Graphs).
The category **Graphs** has the class of graphs as objects. Its morphisms are all graph morphisms, with componentwise composition and identities.

By means of graph morphisms, graphs can be typed, e.g. one can define which type of nodes can be the source or target of which type of edges and so on. Therefore a type graph defines a set of types. To assign a type to the nodes and the edges of a graph a morphism to the type graph is used.

Definition 4 (Typed Graphs and Typed Graph Morphism).
A *type graph* is a distinguished graph $TG = (V_{TG}, E_{TG}, s_{TG}, t_{TG})$. V_{TG} and E_{TG} are called the vertex and the edge type alphabets, respectively.

A tuple $(G, type)$ of a graph G together with a graph morphism type: $G \to TG$ is then called a *typed graph*.

Given typed graphs $G_1^T = (G_1, type_1)$ and $G_2^T = (G_2, type_2)$, a *typed graph morphism* $f : G_1^T \to G_2^T$ is a graph morphism $f : G_1 \to G_2$ such that $type_2 \circ f = type_1$



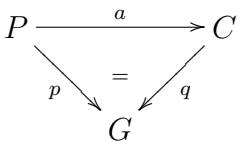
Typed graphs lead to another category $\mathbf{Graphs}_{TG}$.

Definition 5 (The Category $\mathbf{Graphs}_{TG}$).
Given a type graph TG, typed graphs over TG and typed graph morphisms form the category $\mathbf{Graphs}_{TG}$.

Properties for graphs can be formulated with graph constraints. They specify the condition that a graph G must or must not contain a certain subgraph G'. Furhtermore, if a graph contains a certain premise P, we can require that it contains a certain conclusion C.

Definition 6 (Graph Constraint).
An *atomic graph constraint* is of the form *true* or $PC(a)$, where $PC(a) : P \xrightarrow{a} C$is a graph morphism.

A *graph constraint* is a Boolean formula over atomic graph constraints. This means that true and every atomic graph constraint are graph constraints, and, for graph constraints c and c_i with $i \in I$ for some finite index set I, $\neg c$, $\wedge_{i \in I} c_i$, and $\vee_{i \in I} c_i$ are graph constraints:



A graph G *satisfies* a graph constraint c, written $G \models c$, if

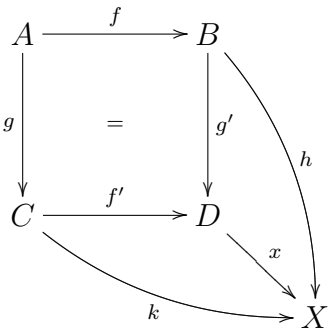
- $c = true$;
- $c = PC(a)$ and, fo every injective graph morphism $p : P \to G$, there exists an injective graph morphism $q : C \to G$ such that $q \circ a = p$;
- $c = \neg c'$ and G does not satisfy c';
- $c = \wedge_{i \in I} c_i$ and G satisfies all c_i with $i \in I$;
- $c = \vee_{i \in I} c_i$ and G satisfies some c_i with $i \in I$;

For the rule based model transformation a technique is needed, which leads to the generation of a new graph by the use of given graphs. Therefore we use the idea of a pushout from category theory, where a pushout object emerges from gluing two objects along a common subobject. In the context of the category of graphs, this technique is called pushout, which enables us to glue two graphs together along a common subgraph. Intuitively, this means, we use this common subgraph and add all other nodes and edges from both graphs.

Definition 7 (Pushout).
Given morphisms $f : A \to B$ and $g : A \to C$ in a category $\mathbf{C}$, a *pushout* (D, f', g') over f and g is defined by

- a pushout object D and
- morphisms $f' : C \to D$ and $g' : B \to D$ with $f' \circ g = g' \circ f$

such that the following universal property is fulfilled: For all objects X and morphisms $h : B \to X$ and $k : C \to X$ with $k \circ g = h \circ f$, there is a unique morphism $x : D \to X$ such that $x \circ g' = h$ and $x \circ f' = k$:



We shall often use the abbrevation “PO” for “pushout”.

Example 1 (Pushout in Category Graphs).
Given the graphs $G1$, $G2$ and $G3$ in Figure 1 with the morphisms depicted by the morphisms from $G1$ to $G3$ by the node mappings. In $G3$ the nodes a and b of $G1$ are glued together. Hence, by the morphism from $G2$ to the pushout object $G4$ these nodes are glued together as well. Note that the diagram in Figure 1 commutes and has the universal property required in Definition 7.

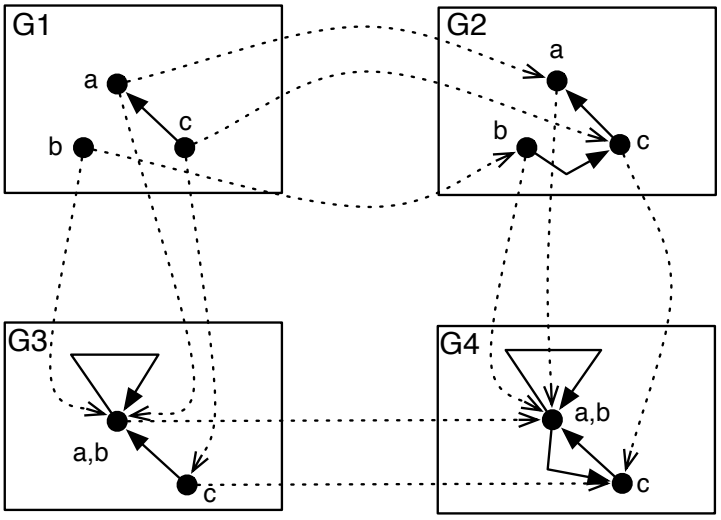


Figure 1: Example for a pushout in **Graphs**

Additionally we will need some properties of pushouts, which are very important for the theory of algebraic graph transformation.

Fact 1 (uniqueness, composition, and decomposition of POs).
Given a category **C**, we have the following:

1. The pushout object D is unique up to isomorphism.
2. The composition and decomposition of pushouts result again in a pushout, i.e., given the following commutative diagram, the statements below are valid:

$$\begin{array}{ccccc} A & \xrightarrow{f} & B & \xrightarrow{e} & E \\ {\scriptstyle g}\downarrow & (1) & \downarrow{\scriptstyle g'} & (2) & \downarrow{\scriptstyle e''} \\ C & \xrightarrow{f'} & D & \xrightarrow{e'} & F \end{array}$$

- Pushout composition: If (1) and (2) are pushouts, then (1) + (2) is also a pushout.
- Pushout decompostion: If (1) and (1) + (2) are pushouts, then (2) is also a pushout.

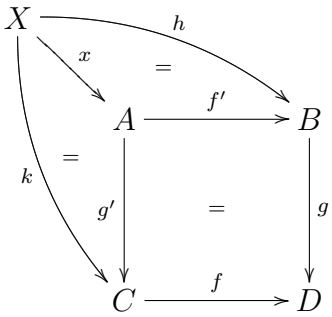
Proof see [EEPT06].

The dual construction of pushouts are pullbacks in category theory. Intuitively the pushout can be seen as a union of two objects over a common object. The pullback can be seen as an intersection of subobjects of a common object.

Definition 8 (Pullback).
Given morphisms $f : C \to D$ and $g : B \to D$ in a category $\mathbf{C}$, a *pushout* (A, f', g') over f and g is defined by

- a pullback object A and
- morphisms $f' : A \to B$ and $g' : A \to C$ with $g \circ f' = f \circ g'$

such that the following universal property is fulfilled: For all objects X and morphisms $h : X \to B$ and $k : X \to C$ with $f \circ k = g \circ h$, there is a unique morphism $x : X \to A$ such that $f' \circ x = h$ and $g' \circ x = k$:



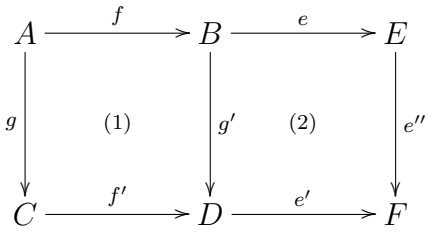
We shall often use the abbrevation "PB" for "pullback".

According to the duality principle in category theory dually to uniqueness, composition, and decomposition of pushouts described in Fact 1, there are corresponding properties for pullbacks in Fact 2.

Fact 2 (Uniqueness, Composition, and Decomposition of PBs).
Given a category $\mathbf{C}$, we have the following:

1. The pullback object A is unique up to isomorphism.

2. The composition and decomposition of pushouts result again in a pushout, i.e., given the following commutative diagram, the statements below are valid:

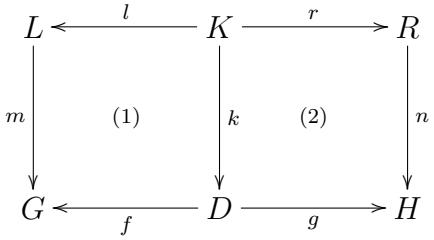


- Pullback composition: If (1) and (2) are pulbacks, then (1) + (2) is also a pullback.
- Pullback decompostion: If (1) and (1) + (2) are pullbacks, then (2) is also a pullback.

Based on the above constructions, especially pushouts, we will now look at the techniques of graph transformation. Graph productions are the basis for algebraic graph transformation. They describe a general way how to transform certain patterns in a graph. If the conditions needed for the application of the rule are satisfied, the pattern can be transformed, resulting in a *direct graph transformation*. In the following, the concepts of graph and typed graph transformation systems will be handeled both simultaneously. Therefore we will use the abbreviated terminology, (typed) graph transformation systems.

Definition 9 (Graph Production).
A *(typed) graph production* $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ consists of (typed) graphs L, K and R, called the left-hand side, gluing graph, and the right-hand side respectively, and two injective (typed) graph morphisms l and r.

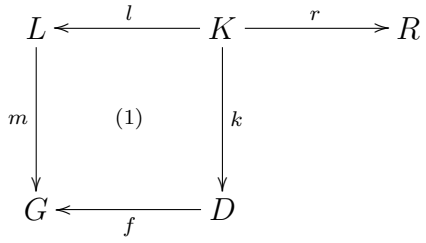
Definition 10 (Graph Transformation).
Given a (typed) graph production $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ and a (typed) graph G with a (typed) graph morphism $m : L \to G$, called the match, a *direct (typed) graph transformation* $G \overset{p,m}{\Longrightarrow} H$ from G to a (typed) graph H is given by the following double-pushout (DPO) diagram, where (1) and (2) are pushouts in the category **Graphs** (or $\mathbf{Graphs_{TG}}$, respectively):



A sequence $G_0 \Rightarrow G_1 \Rightarrow ... \Rightarrow G_n$ of direct (typed) graph transformations is called a *(typed) graph transformation* and is denoted by $G_0 \Rightarrow^* G_n$. For $n = 0$, we have the identical (typed) graph transformation $G_0 \overset{id}{\Longrightarrow} G_0$. Moreover, for $n = 0$ we allow also graph isomorphisms $G_0 \cong G_0'$, because pushouts and hence also direct graph transformations are only unique up to isomorphism.

To apply a (typed) graph production $p = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ to a (typed) graph G via a match m certain conditions have to be fulfilled. Given $l : K \rightarrow L$ and $m : L \rightarrow G$ we hve to make sure the existence of a context graph D such that G becomes a pushout of L and D over K. The existence of such a context graph, that leads to the first pushout, allows us to construct the (typed) graph H in a second step as the pushout of D and R over K. This is the intuitive introduction to the construction of the direct (typed) graph transformation $G \overset{p,m}{\Longrightarrow} H$.

Definition 11 (Applicability of a Production).
A (typed) graph production $p = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ is applicable to a (typed) graph G via the match m if there exists a context graph D such that (1) is a pushout in the sense of Definition 10.



Since this definition is a rather abstract view from the category theoretical point of view, it gives no syntactical criterion to decide wheter a (typed) graph production is applicable or not. Such a criterion is formulated by the gluing condition, which is necessary and sufficiant for the existence of a context graph.

Definition 12 (Gluing Condition).
Given a (typed) graph production $p = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$, a (typed) graph G, and a match $m : L \rightarrow G$ with $X = (V_X, E_X, s_X, t_X)$ for all $X \in \{L, K, R, G\}$, we can state the following definitions:

- The *gluing points* GP are those nodes and edges in L that are not deleted by p, i.e $GP = l_V(V_K) \cup l_E(E_K) = l(K)$.
- The *identification points* IP are those nodes and edges in L that are identified by m, i.e. $IP = \{v \in V_L \mid \exists w \in V_L, w \neq v : m_V(v) = m_V(w)\} \cup \{e \in E_L \mid \exists f \in E_L, f \neq e : m_E(e) = m_E(f)\}$.

- The *dangling points* DP are those nodes in L whose images under m are the source or target of an edge in G that does not belong to $m(L)$, i.e. $DP = \{v \in V_L \mid \exists e \in E_G \setminus m_E(E_L) : s_G(e) = m_V(v) \vee t_G(e) = m_V(v)\}$.

p and m satisfy the *gluing condition* if all identification points and all dangling points are also gluing points, i.e. $IP \cup DP \subseteq GP$.

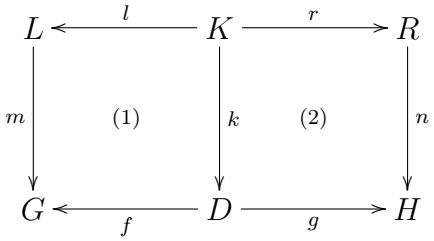
If the gluing condition is not satisfied, the production p cannot be applied via the match m. But if it is applicable, then the direct (typed) graph transformation can be constructed as follows.

Fact 3 (Construction of Direct Graph Transformations).
Given a (typed) graph production $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ and a match $m : L \to G$ such that p is applicable to a (typed) graph G via m, the direct (typed) graph transformation $G \overset{p,m}{\Longrightarrow} H$ can be constructed in two steps:

1. Delete those nodes and edges in G that are reached by the match m, but keep the image of K, i.e. $D = (G \setminus m(L)) \cup m(l(K))$. More precisely, construct the context graph D and pushout (1) such that $G = L +_K D$.

2. Add those nodes and edges that are newly created in R, i.e. $H = D \uplus (R \setminus r(K))$, where the disjoint union $\uplus$ is used to make sure that we add the elements of $R \setminus r(K)$ as new elements. More precisely, construct the pushout (2) of D and R via K such that $H = R +_K D$.

The constructions in step 1 and 2 are unique up to isomorphism.



Example 2 (Direct Graph Transformation Step).
A simple example of a DPO graph transformation step is given in Figure 2. The letters next to the nodes indicate how they are mapped by graph morphisms. Note that the gluing condition is satisfied, because the dangling point a of L is also gluing point. Therefore, the context graph C can be constructed, such that (1) is a pushout over L and C along K in the category **Graphs**. Moreover, H is the pushout object of the pushout over C and R along K.

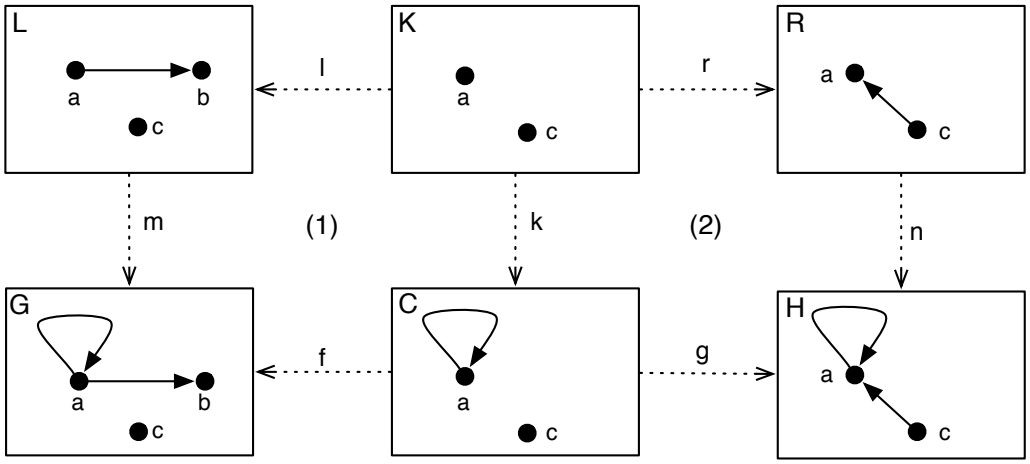
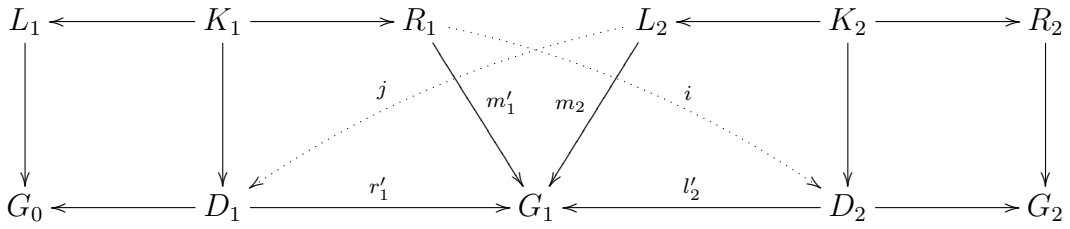


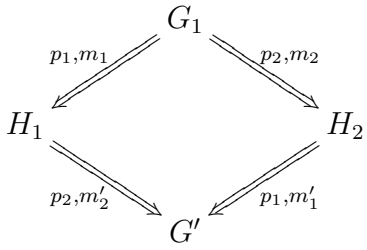
Figure 2: Example for a DPO transformation step

Under certain conditions two direct (typed) graph transformations applied to the same (typed) graph can be applied in arbitrary order, leading to the same result. Here we use the notion of sequential independendence of direct (typed) graph transformations and the Local Church-Rosser Theorem.

Definition 13 (Sequential Independence).
Let $d = (G_0 \overset{p_1,m_1}{\Longrightarrow} G_1 \overset{p_2,m_2}{\Longrightarrow} G_2)$ be a sequence of graph transformations . Then, $d_1 = G_0 \overset{p_1,m_1}{\Longrightarrow} G_1$ and $d_2 = G_1 \overset{p_2,m_2}{\Longrightarrow} G_2$ are two sequentially independent transformation steps, if there exist $i : R_1 \to D_2$, $j : L_2 \to D_1$ in the diagram beneath, s.t. $l' \circ i = m'_1$ and $r' \circ j = m_2$.



Theorem 1 (Local Church-Rosser).
Two sequentially independent derivation steps can be switched by the Local Church Rosser Theorem (Thm. 5.12 in [EEPT06]) leading to the following diagram of direct transformations:



The definition of sequential independence and the Local Church-Rosser Theorem leads to the definition of switch equivalence, which enables us to switch derivation steps in a derivation.

Definition 14 (Switch Equivalence).
Let $d = (d_1; ...; d_k; d_{k+1}; ...d_n)$ be a derivation in grammar GG, where $d_k; d_{k+1}$ are two sequentially independent derivation steps. Let d' be derived from d by replacing $(d_k; d_{k+1})$ by $(d'_{k+1}; d'_k)$ according to the Local Church Rosser Theorem. Then, d' is a switching of d, written $d \overset{sw}{\sim} d'$. Switch-equivalence $\overset{sw}{\approx}$ is the union of the transitive closure of $\overset{sw}{\sim}$ and the relation $\cong$ for isomorphic derivations.

It is essential to determine in which cases the application of a transformation rule to graph in a given graph grammar prevents other rule from being applicable to the graph, e.g., by deleting parts of the graph such that a match cannot be found by another. Therefore we need the notion of critical pairs.

Definition 15 (Critical Pairs).
A *critical pair* is a pair of parallel dependent direct transformations $P1 \overset{p_1,m_1}{\Longleftarrow} K \overset{p_2,m_2}{\Longrightarrow} P_2$ such that $m_1 : L_1 \to K$ and $m_2 : L_2 \to K$ are jointly surjective.

2.2 Triple Graphs

After the introduction to the basics of algebraic graph transformation in the last section, in this section the concept of graphs will be extended to triple graphs in order to define model transformation in the sense of [Sch94]. Triple graphs consist of three graphs, a source, a correspondence, and a target graph. In the concept of model transformation with triple graph grammars the source and the target graphs are models which are connected via the intermediate correspondence graph and suitable graph morphisms.

Definition 16 (Triple Graph).
Three graphs G^S, G^C, and G^T, called source, connection, and target graphs, together with two graph morphisms $s_G : G^C \to G^S$ and $t_G : G^C \to G^T$ form a *triple graph* $G = (G^S \overset{s_G}{\longleftarrow} G^C \overset{t_G}{\longrightarrow} G^T)$. G is called empty, if G^S, G^C, and G^T are empty graphs.

Like graphs, triple graphs can be related to each other by combining three graph morphisms to a triple graph morphisms under the condition, that the morphisms from the correspondence graph are preserved.

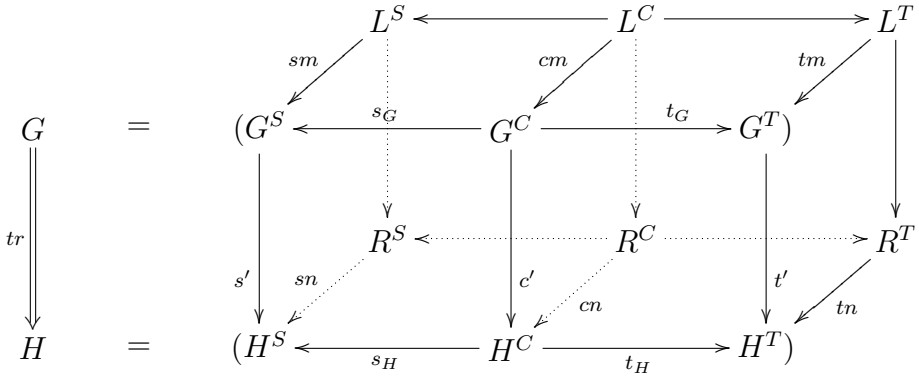
Definition 17 (Triple Graph Morphism).
A *triple graph morphism* $m = (s, c, t) : G \to H$ between two triple graphs $G = (G^S \xleftarrow{s_G} G^C \xrightarrow{t_G} G^T)$ and $H = (H^S \xleftarrow{s_H} H^C \xrightarrow{t_H} H^T)$ consists of three graph morphisms $s : G^S \to H^S$, $c : G^C \to H^C$ and $t : G^T \to H^T$ such that $s \circ s_G = s_H \circ c$ and $t \circ t_G = t_H \circ c$.

The technique of triple graph transformation is based on the same approach as that of graph transformations. The source, the target, and their correspondence graph are build by triple rules, which are non-deleting, i.e., they only consist of a left-hand side and a right-hand side without a gluing graph. Structure deletion to extract only parts of the triple graphs is not necessary for triple rule applications. This avoids to check the gluing condition for triple rule transformation steps.

Definition 18 (Triple Rule).
A *triple rule* tr consists of triple graphs L and R, called left-hand and right-hand sides, and an injective triple graph morphism $tr = (s, c, t) : L \to R$.

$$\begin{array}{ccc} L & = & (L^S \xleftarrow{s_L} L^C \xrightarrow{t_L} L^T) \\ {\scriptstyle tr}\downarrow & & \downarrow{\scriptstyle s} \quad \downarrow{\scriptstyle c} \quad \downarrow{\scriptstyle t} \\ R & = & (R^S \xleftarrow[s_R]{} R^C \xrightarrow[t_R]{} R^T) \end{array}$$

Definition 19 (Triple Transformation Step).
Given a triple rule $tr = (s, c, t) : L \to R$, a triple graph G and an injective triple graph morphism $m = (sm, cm, tm) : L \to G$, called triple match m, a triple graph transformation step (TGT-step) $G \overset{tr,m}{\Longrightarrow} H$ from G to a triple graph H is given by three pushouts (H^S, s', s_n), (H^C, c', c_n) and (H^T, t', t_n) in the category **Graph** with induced morphisms $s_H : H^C \to H^S$ and $t_H : H^C \to H^T$. Morphism $n = (sn, cn, tn)$ is called comatch.



A sequence of triple graph transformation steps is called triple (graph) transformation sequence, short: TGT-sequence. Moreover, we obtained a triple graph morphism $d : G \to H$ with $d = (s', c', t')$, called transformation morphism.

With triple rules triple languages can be defined, with their source and target language, which are defined by projection to the source and target graphs respectively. This has been defined in [EEH08]

Definition 20 (Triple, Source and Target Language).
A set of triple rules TR defines the *triple language* $VL = \{G \mid \emptyset \Rightarrow G \text{ via } TR\}$ of triple graphs. *Source language* VL_S and *target language* VL_T are derived by projection to the triple components, i.e. $VL_S = proj_S(VL)$ and $VL_T = proj_T(VL)$, where $proj_X$ is a projection defined by restriction to one of the triple components, i.e. $X \in \{S, C, T\}$. The languages generated by the source rules TR_S and target rules TR_F are denoted by VL_{S0} and VL_{T0} respectively, where $VL_S \subseteq VL_{S0}$ and $VL_T \subseteq VL_{T0}$.

For the creation and transformation of models in triple grammars, certain rules can be derived from a given triple rule, which are only creating in one domain, either based on empty models (source and target rules) or given models (forward and backward rules) in the respective domain.

Definition 21 (Derived Triple Rules).
From each triple rule $tr = L \to R$ as given in Definition 18 there are the following source, target, forward and backward rules:

$$\begin{array}{ccc} (L^S & \longleftarrow \emptyset \longrightarrow & \emptyset) \\ {\scriptstyle s}\downarrow & \downarrow & \downarrow \\ (R^S & \longleftarrow \emptyset \longrightarrow & \emptyset) \end{array} \qquad \begin{array}{ccc} (\emptyset & \longleftarrow \emptyset \longrightarrow & L^T) \\ \downarrow & \downarrow & \downarrow{\scriptstyle t} \\ (\emptyset & \longleftarrow \emptyset \longrightarrow & R^T) \end{array}$$

source rule tr_S target rule tr_T

$$\begin{array}{ccccc} (R^S & \xleftarrow{s \circ s_L} & L_C & \xrightarrow{t_L} & L^T) \\ {\scriptstyle id}\downarrow & & \downarrow{\scriptstyle c} & & \downarrow{\scriptstyle t} \\ (R^S & \xleftarrow{s_R} & R^C & \xrightarrow{t_R} & R^T) \end{array} \qquad \begin{array}{ccccc} (R^S & \xleftarrow{s \circ s_L} & L_C & \xrightarrow{t_L} & L^T) \\ {\scriptstyle id}\downarrow & & \downarrow{\scriptstyle c} & & \downarrow{\scriptstyle t} \\ (R^S & \xleftarrow{s_R} & R^C & \xrightarrow{t_R} & R^T) \end{array}$$

forward rule tr_F backward rule tr_B

All elements of the source model language can be created with source rules. Since they contain less restrictions for matches during transformation in comparison to their corresponding complete triple rule, they possibly allow to generate more elements than the source model language contains. From an existing source model a target model can be derived with a sequence of forward rules.

In order to have correct and complete transformations based on triple graph grammars we need the notion of match and source consistency from [EEHP09]. Thus, we can check, if a forward sequence has a corresponding source sequence, such that the source and target

model are derived from the same triple rule sequence.

Definition 22 (Match and Source Consistency).
Let tr_S^* and tr_F^* be sequences of source rules tri_S and forward rules tri_F, which are derived from the same triple rules tri for $i = 1, ..., n$. Let further $G_{00} \overset{tr_S^*}{\Longrightarrow} G_{n0} \overset{tr_F^*}{\Longrightarrow} G_{nn}$ be a TGT-sequence with (mi_S, ni_S) being match and comatch of tri_S (respectively (mi_F, ni_F) for tri_F) then match consistency of $G_{00} \overset{tr_S^*}{\Longrightarrow} G_{n0} \overset{tr_F^*}{\Longrightarrow} G_{nn}$ means that the S-component of the match mi is uniquely determined by the comatch ni_S $(i = 1, ..., n)$. A TGT-sequence $G_{n0} \overset{tr_F^*}{\Longrightarrow} G_{nn}$ is source consistent, if there is a match consistent sequence $\emptyset \overset{tr_S^*}{\Longrightarrow} G_{n0} \overset{tr_F^*}{\Longrightarrow} G_{nn}$. Note that by source sonsistency the application of the forward rules is controlled by the source sequence, which generates the given source model.

For the notion of match and source consistency is is essential that each TGT-sequence can be decomposed in transformation sequences by corredponding source and forward rules and vice versa, provided that their matches are consistent (see [EEE+07]).

Theorem 2 (Decomposition and Composition of TGT-Sequences).
Given triple rules tr_i with source rule $tr_{i,S}$ and forward rule $tr_{i,F}$ for $i = 1, ..., n$:

1. **Decomposition:** For each TGT-sequence
$$G_0 \overset{tr_1}{\Longrightarrow} G_1 \Rightarrow ... \overset{tr_n}{\Longrightarrow} G_n \quad (1)$$
there is a corresponding match consistent TGT-sequence
$$G_0 = G_{00} \overset{tr_{1S}}{\Longrightarrow} G_{10} \Rightarrow ... \overset{tr_{nS}}{\Longrightarrow} G_{n0} \overset{tr_{1F}}{\Longrightarrow} G_{n1} \Rightarrow ... \overset{tr_{nF}}{\Longrightarrow} G_{nn} = G_n \quad (2)$$

2. **Composition:** For each match consistent transformation sequence (2) there is a canonical transformation sequence (1).

3. **Bijective Correspondence:** Composition and decomposition are inverse to each other.

Definition 23 (Model Transformation).
$(G_S, G \overset{tr_F^*}{\Longrightarrow} H, H_T)$ is a model transformation sequence from G_S to H_T , if $G \overset{tr_F^*}{\Longrightarrow} H$ is source consistent, where G_S and H_T are the source and target graphs of G and H, respectively. A *model transformation* $MT : VL_{S0} \Rrightarrow VL_{T0}$ consists of model transformation sequences $(G_S, G \overset{tr_F^*}{\Longrightarrow} H, H_T)$ with $G_S \in VL_{S0}$, $H_T \in VL_{T0}$.

It is essential to determine, if a model transformation generates correct and complete models in the given language. Furthermore, it is important, whether the transformation is terminating. In [**?**] this has been defined.

Theorem 3 (Correctness, Completeness and Termination).
Each model transformation $MT : VL_{S0} \Rrightarrow VL_{T0}$ based on forward translation rules is

1. **correct**, i.e. for each model transformation sequence $(G_S, G_0 \overset{tr_F^*}{\Longrightarrow} G_n, G_T)$ there is $G \in VL$ with $G = (G_S \leftarrow G_C \rightarrow G_T)$, and hence $G_S \in VL_S$ and $G_T \in VL_T$.
2. **complete**, i.e. for each $G_S \in VL_S$ there is $G = (G_S \leftarrow G_C \rightarrow G_T) \in VL$ with a model transformation sequence $(G_S, G_0 \overset{tr_F^*}{\Longrightarrow} G_n, G_T)$.
3. **terminating**, i.e. for a source model $G_S \in VL_S$ (target model $G_T \in VL_T$) and a set of triple rules such that G_S (G_T) and all rule components are finite on the graph part and the triple rules are creating on the source (target) component. Then each model transformation sequence $(G_S, G_0 \overset{tr_F^*}{\Longrightarrow} G_n, G_T)$ is terminating, i.e. any extended sequence $G_0 \overset{tr_F^*}{\Longrightarrow} G_n \overset{tr_F'^*}{\Longrightarrow} G_m$ is not source consistent.

3 Model Transformation with Borrowed Context

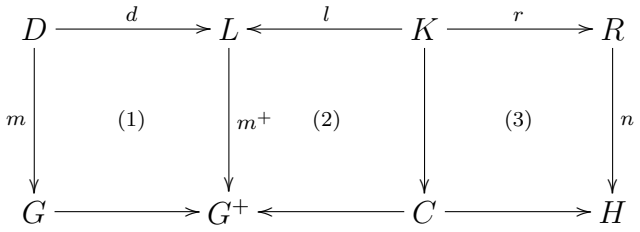
After introducing model transformation using triple graph grammars in the previous section in this section a new approach for transforming model fragments using triple graph grammars with borrowed context will be introduced. This is motivated by propagation of graph constraints in Section 4, where premise and conclusion of graph constraints are in general only model fragments. In Section *3.1 Definition* we will define the notation of the borrowed context. Then we will define source consistency for borrowed context and we willshow under which conditions the borrowed context can be extended to a standard DPO-transformation in section *3.2 Extension of Borrowed Context Transformations and Source Consistency*. In section *3.3 Correctness, Completeness and Termination* we will define under which conditions a graph transformation system with borrowed context transformations is correct, complete and terminating.

3.1 DPO-Transformation with Borrowed Context

For a standard algebraic graph transformation of a graph G with a transformation rule $p = (L \leftarrow K \rightarrow R)$ total graph morphisms are required as matches $m : L \rightarrow G$. The borrowed context approach allows to apply a transformation rule also with partial matches $m : L \rightharpoonup G$. The idea behind this concept is to "borrow" the parts that are not in the domain of the partial match. This will be done by gluing the graphs L and G over the common part given by the injective match. Therewith we get a new graph G^+ and a total match $m^+ : L \rightarrow G^+$ so that a standard DPO-transformation step can be executed.

Definition 24 (BC-Transformations).
Given a DPO-Rule $p = (L \leftarrow K \rightarrow R)$ and a Graph G with partial match $m^- : L \rightharpoonup G$ given by an injective span $(L \overset{d}{\leftarrow} D \overset{m}{\rightarrow} G)$ then the total match $m^+ : L \rightarrow G^+$ is defined

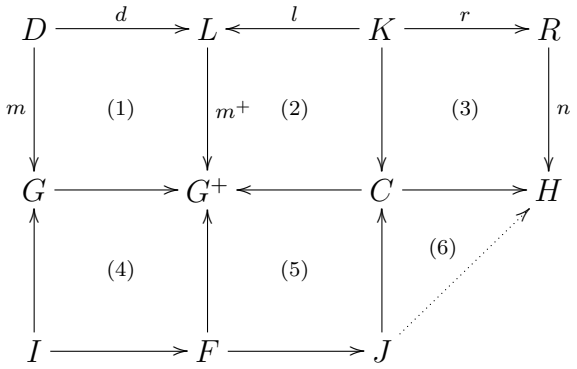
by $PO(1)$ and a BC-transformation step $G \overset{p,d,m}{\Longrightarrow}_{BC} H$, short $G \overset{p,m}{\Longrightarrow}_{BC} H$ is given by POs (1)-(3), which includes a standard DPO-transformation step $G^+ \overset{p,m^+}{\Longrightarrow} H$ given by POs (2) and (3).



A BC-transformation is a sequence of BC-transformation steps.

$$G_0 \Rightarrow^*_{BC} G_n = G_0 \overset{p_1,m_1}{\Longrightarrow}_{BC} G_1 \overset{p_2,m_2}{\Longrightarrow}_{BC} \ldots \overset{p_{n-1},m_{n-1}}{\Longrightarrow}_{BC} G_n$$

Remark
Intuitively the borrowed context is given by $G^+ \setminus G$ which is given by the graph F in $PO(4)$, where G is given together with an interface $I \to G$ and the interface J is constructed by $PB(5)$ leading by composition in (6) to an interface $J \to H$. Alltogether we obtain a BC-transformation step for graphs with interfaces in the sense of [EK04], written $(G, I) \overset{p,m,(I \leftarrow F \rightarrow J)}{\Longrightarrow} (H, J)$, which is defined by $POs(1) - (4)$ and $PB(5)$, where C in (2) and F in (4) have to be constructed as PO-complement, provided that the corresponding gluing conditions are satisfied.



Example 3
In the following, Figure 3 shows an example for a BC-transformation step in a plain graph. There exists no total injective match from the LHS of the rule L, because the node c can not be matched. Hence, there exists only the partial match $m^- : L \rightharpoonup G$, matching the nodes a and b and the edge and we get the graph D and morphisms d and m as injective span. The pushout over this span gives us G^+ and a total match match $m^+ : L \to G^+$. Finally, the DPO-transformation step can be executed, giving us as a result the graph H.

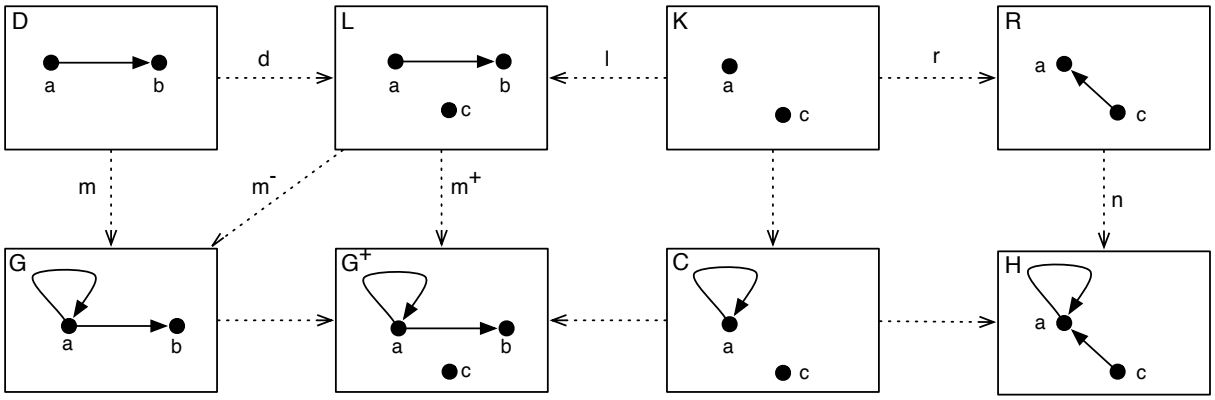
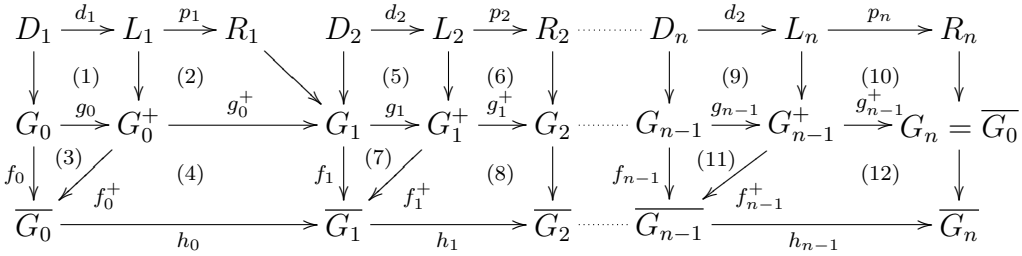


Figure 3: Example for a DPO-transformation with BC

Given a *BC-transformation* $G_0 \Rightarrow^*_{BC} G_n$ via $(p_1, ...p_n)$ it is interesting to find out under which conditions this *BC-transformation* can be extended to a standard DPO-transformation $\overline{G_0} \Rightarrow^* \overline{G_n}$ via $(p_1, ...p_n)$, where $G_0 \to \overline{G_0}$ is a suitable extension. This problem is closely related to the Embedding and Extension Theorems. In our case $\overline{G_0}$ might be constructed as colimit of all the diagrams of $G_0 \Rightarrow^*_{BC} G_n$, but we would need suitable consistency conditions. The problem seems to be easier if all rules $p_1, ..., p_n$ are non-deleting. In this case the colimit is G_n and we could take $\overline{G_0} = G_n$. It would be interesting to find morphisms $f_0, f_0^+, f_1, f_1^+, ..., f_{n-1}, f_{n-1}^+$ such that $(3), (7), ..., (11)$ commutes and $(4), (8), ..., (12)$ are POs.



This is easy for $f_0^+ = g_n^+ \circ g_{n-1} \circ ... \circ g_1^+ \circ g_1 \circ g_0^+$ and $f_0 = f_0^+ \circ g_0$ such that (3) commutes, and to define f_1 by $PO(4)$. But it is already non-trivial to find f_1^+ such that (7) commutes etc.

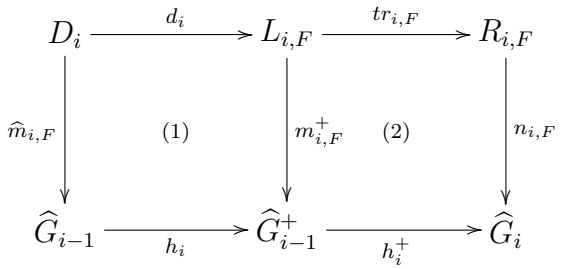
Later we will study this situation for triple graphs with BC-forward sequences, where we have the special case that in the S-components $p_1, p_2, ..., p_n$ are identities and hence also $g_0^+, g_1^+, ..., g_{n-1}^+$ are identities in the S-component. In the C- and T-components $d_1, d_2, ..., d_n$ are identities leading to identities in the C- and T-components for $g_0, g_1, ..., g_{n-1}$. There are straight forward constructions for the S-components on one hand and C- and T-components of $f_1, f_1^+, ..., f_{n-1}, f_{n-1}^+$ on the other hand and it essentially remains to show, that they are triple graph morphisms (see Theorem 5 below).

3.2 Extension of Borrowed Context Transformations and Source Consistency

After defining in Section 3.1 the DPO-transformation with borrowed context, in this section we will introduce the notion of forward transformation in triple graph grammars. Furhtermore, we will show, that such a transformation is consistent and that can be extended to a forward transformation without borrowed context.

The idea of a forward transformation with borrowed context is to borrow only on the source components, while the rule only makes changes to the correspondence and target components.

Definition 25 (Forward Transformation with BC).
Given triple rules TR with corresponding forward rules TR_F then a *BC-forward transformation* $\widehat{G}_0 \Rightarrow^*_{BC} \widehat{G}_n$ via TR_F is given by BC-forward transformation steps $\widehat{G}_{i-1} \overset{tr_{i,F},\widehat{m}_{i,F},d_i}{\Longrightarrow} \widehat{G}_i$



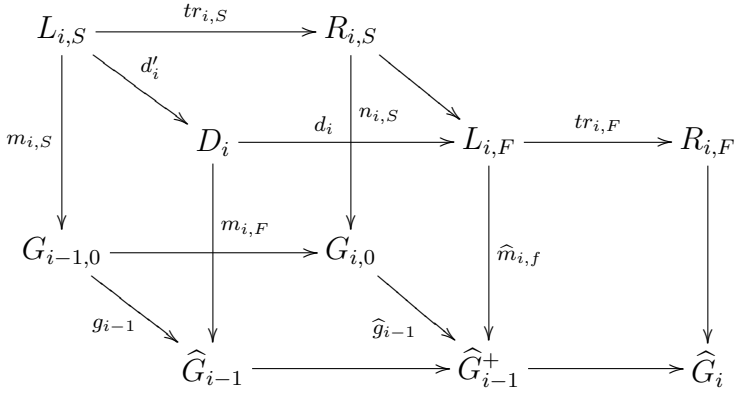
for $i = 1, ..., n$ with POs (1) and (2) of triple graphs with injective matches $\widehat{m}_{i,F}$, where the C- and T-components $d_{i,C}$ and $d_{i,T}$ of d_i are identities. Moreover, the S-component $tr_{i,F,S}$ is the identity of $R_{i,S}$ by construction of TR_F, i.e. $tr_{i,F,S} = id_{R_{i,S}}$.

Furthermore, the trace of a BC-forward transformation step i is given by $trace(\widehat{G}_{i-1} \overset{tr_{i,F},\widehat{m}_{i,F},d_i}{\Longrightarrow} \widehat{G}_i) = h^+_i \circ h_i$ and the trace of a BC-forward transformation is given by $trace(\widehat{G}_0 \overset{tr^*_F}{\Longrightarrow}_{BC} \widehat{G}_n) = h^+_n \circ h_n \circ ... \circ h^+_1 \circ h_1$.

We have to define under which conditions a forward transformation with borrowed context is source consistent (see Definition 22), i.e., that there exists a sequence of source rules, corresponding to the forward rules, which generates the model in the source component after borrowing.

Definition 26 (Partial BC-Match and BC-Source Consistency).
A pair $(G_{00} \overset{tr^*_S}{\Longrightarrow} G_{n0}, \widehat{G}_0 \overset{tr^*_F}{\Longrightarrow}_{BC} \widehat{G}_n)$ of a source sequence $G_{00} \overset{tr^*_S}{\Longrightarrow} G_{n0}$ and a BC-forward transformation $\widehat{G}_0 \overset{tr^*_F}{\Longrightarrow}_{BC} \widehat{G}_n$ with corresponding forward rules tr^*_F defined by POs (1), (2) and (3) respectively for $i = 1, ..., n$ with $G_{00} = \emptyset, \widehat{G}^C_0 = \widehat{G}^T_0 = \emptyset$ is called *partially BC-match consistent*, if we have for $i = 1, ..., n$ inclusions d'_i, g_{i-1}, and $\widehat{g}_{i-1}$, such that using

the inclusion $R_{i,S} \hookrightarrow L_{i,F}$ the cube commutes and left face is PB, where (1) and (2) are the POs in the front and (3) PO in the back face with inclusion $G_{i,0} \subseteq \widehat{G}_i$. The pair $(G_{00} \overset{tr_S^*}{\Longrightarrow} G_{n0}, \widehat{G}_0 \overset{tr_F^*}{\Longrightarrow}_{BC} \widehat{G}_n)$ is called *total BC-match consistent*, if we have in addition $G_{n,0}^S = \widehat{G}_n^S$

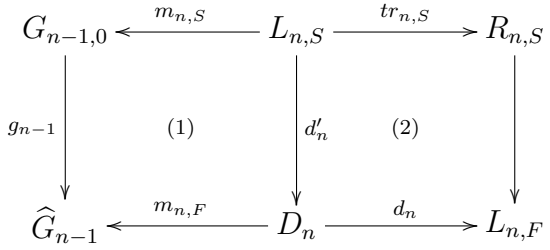


A BC-forward sequence $\widehat{G}_0 \overset{tr_F^*}{\Longrightarrow}_{BC} \widehat{G}_n$ is called *BC-source consistent*, if there is a corresponding source sequence $G_{00} \overset{tr_S^*}{\Longrightarrow} G_{n0}$, such that both together are total BC-match consistent.

Remark
Commutativity of the right face shows that the S-component of the extended match $\widehat{m}_{i,F}$ is compatible with the comatch $n_{i,S}$. The PB-property of the left means $m_{i,F}^{-1}(G_{i-1,0}) = L_{i,S}$

In order to continue a source consistent forward sequence with another forward transformation step such that the resulting forward sequence is source consistent, we need to define when a BC-match is forward consistent.

Definition 27 (Forward Consistent BC-Match).
Given a partially BC-match consistent pair $(G_{00} \overset{tr_S^*}{\Longrightarrow} G_{n-1,0}, \widehat{G}_0 \overset{tr_F^*}{\Longrightarrow}_{BC} \widehat{G}_{n-1})$ with $g_{n-1} : G_{n-1,0} \to \widehat{G}_{n-1}$ then a partial match $(L_{n,F} \overset{d_n}{\hookleftarrow} D_n \overset{m_{n,F}}{\longrightarrow} \widehat{G}_{n-1})$ for $tr_{n,F} : L_{n,F} \to R_{n,F}$ with $tr_{n,S} : L_{n,S} \to R_{n,S}$ is called *forward consistent BC-match*, if there is a source match $m_{n,S} : L_{n,S} \to G_{n-1,0}$ and an inclusion $d'_n : L_{n,S} \hookrightarrow D_n$, such that (1) is PB and (2) commutes.



Remark
PB (1) means $m_{n,F}^{-1}(G_{n-1,0}) = L_{n,S}$

Example 4 (Forward Consistent BC-Match).
Given the partially BC-match consistent pair $(G_{00} \overset{tr_S^*}{\Longrightarrow} G_{n-1,0}, \widehat{G}_0 \overset{tr_F^*}{\Longrightarrow}_{BC} \widehat{G}_{n-1})$ shown in Figure 4. On top is the source sequence and in the bottom the corresponding forward sequence.

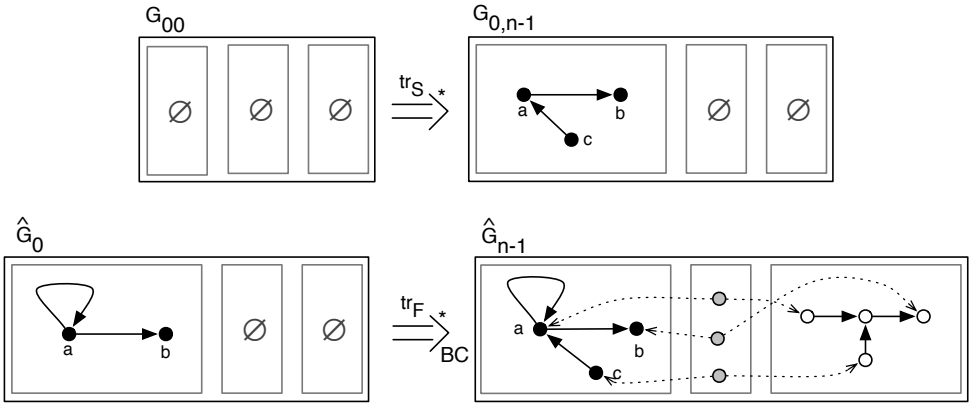


Figure 4: Partially BC-Match Consistent Pair

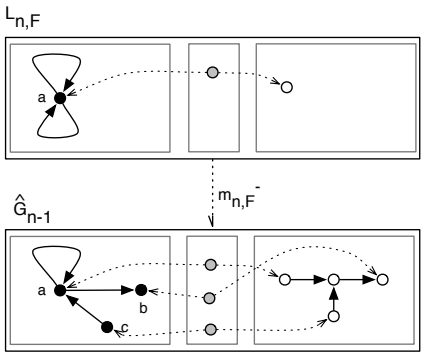


Figure 5: Forward Consistent BC-Match

Now we want to confirm, if the partial match $m_{n,F}^{-} : L_{n,F} \rightharpoonup \widehat{G}_{n-1}$ shown in Figure 6 is a forward consistent BC-match. Therefore we need the source rule $tr_{n,S} : L_{n,S} \to R_{n,S}$. In Figure 6 is in the bottom the span given by the partial match $m_{n,F}^{-}$ and (1) is a pullback and (2) commutes.

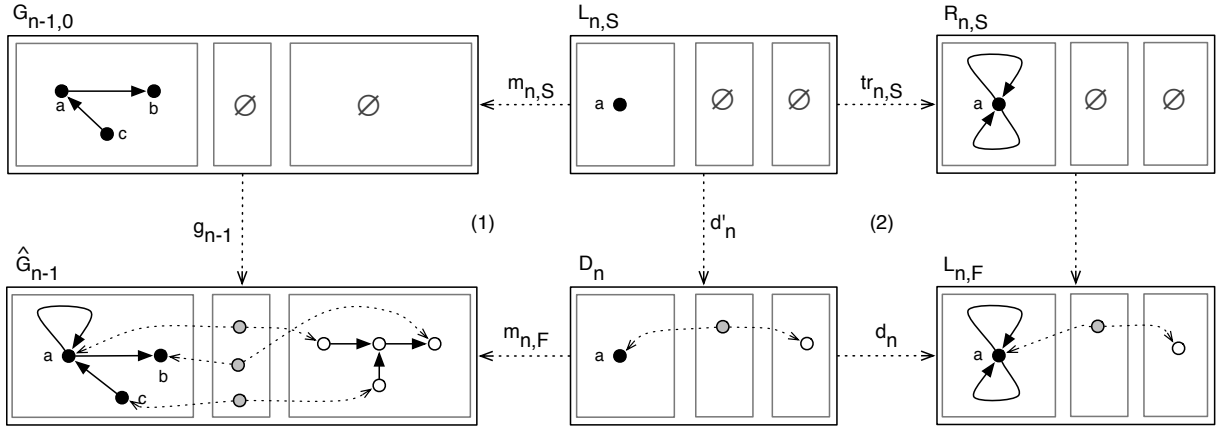


Figure 6: Forward Consistent BC-Match

Since our aim is to execute model transformations using borrowed context, we not only need forward transformation with borrowed context and consistency as seen above, but also a borrowed context model transformation. Now we are able to define a BC-model transformation.

Definition 28 (BC-Model Transformation).
A BC-model transformation sequence $(G_S, \widehat{G}_0 \overset{tr_F^*}{\Longrightarrow}_{BC} \widehat{G}_n, G_T)$ consists of a source consistent BC-forward transformation $\widehat{G}_0 \overset{tr_F^*}{\Longrightarrow}_{BC} \widehat{G}_n$ via TR_F with source model $G_s = \widehat{G}_n^S$ and target model $G_T = \widehat{G}_n^T$. A *BC-model transformation* $MT_{BC} : VL_{S0} \Rrightarrow_{BC} VL_{T0}$ consists of BC-model transformation sequences $(G_S, \widehat{G}_0 \overset{tr_F^*}{\Longrightarrow}_{BC} \widehat{G}_n, G_T)$ with $G_S \in VL_{S0}$ and $G_T \in VL_{T0}$.

After introducing in Definition 26 the partial BC-match and BC-source consistency, which allows us to check after constructing a BC-forward sequence $\widehat{G}_0 \overset{tr_F^*}{\Longrightarrow}_{BC} \widehat{G}_n$, we can check wether a transformation sequence with borrowed context is source consistent. This method has the disadvantage of only being applicable after constructing a forward sequence, which means we first have to construct sequences and possibly to trash them, if we find out afterwards that they are not source consistent. Therefore, we define an on-the-fly construction of a bc-model transformation sequences, which is already source consistent by construction.

Theorem 4 (On-the-Fly Construction of BC-Model Transformation Sequences).
Given a triple graph $\widehat{G}_0$ with $\widehat{G}_0^C = \widehat{G}_0^T = \emptyset$ execute the following steps:

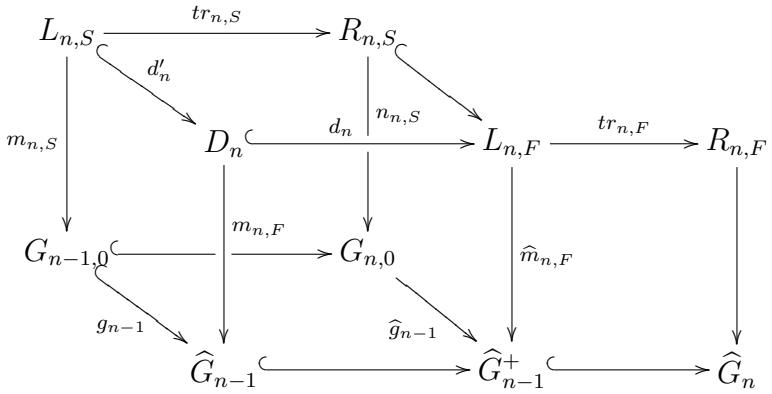
1. Start with $G_{00} = \emptyset$ and inclusion $g_0 : G_{00} \hookrightarrow \widehat{G}_0$.

2. For $n > 0$ and an already computed partial BC-match consistent pair $(G_{00} \overset{tr_S^*}{\Longrightarrow}$

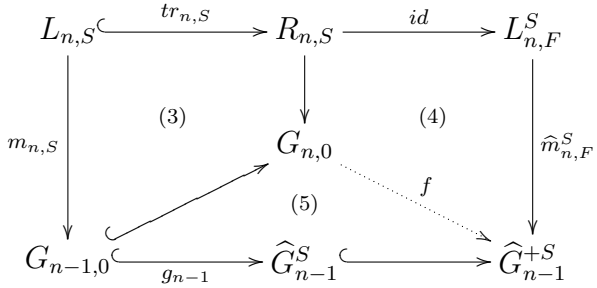
$G_{n-1,0}, \widehat{G}_0 \overset{tr_F^*}{\Longrightarrow}_{BC} \widehat{G}_{n-1}$) with inclusion $g_{n-1} : G_{n-1,0} \hookrightarrow \widehat{G}_{n-1}$ find a rule $tr_{n,F} : L_{n,F} \to R_{n,F}$ together with a partial match $(L_{n,F} \overset{d_n}{\hookleftarrow} D_n \overset{m_{n,F}}{\longrightarrow} \widehat{G}_{n-1})$, which is a forward consistent BC-match, then the given pair can be extended to a partial BC-match consistent pair $(G_{00} \overset{tr_S^*}{\Longrightarrow} G_{n-1,0} \overset{tr_{n,S}}{\Longrightarrow} G_{n,0}, \widehat{G}_0 \overset{tr_F^*}{\Longrightarrow}_{BC} \widehat{G}_{n-1} \overset{tr_{n,F}}{\Longrightarrow}_{BC} \widehat{G}_n)$ with inclusion $g_n : G_{n,0} \hookrightarrow \widehat{G}_n$. Repeat until $G_{n,0} = \widehat{G}_n^S$ or no forward consistent BC-match can be found.

3. If the procedure terminates with $G_{n,0}^S = \widehat{G}_n^S$ then we have a total BC-match consistent pair $(G_{00} \overset{tr_S^*}{\Longrightarrow} G_{n,0}, \widehat{G}_0 \overset{tr_F^*}{\Longrightarrow} \widehat{G}_n)$ with $G_{n,0}^S = \widehat{G}_n^S$ leading to a *BC-model transformation sequence*, i.e. $(G_S, \widehat{G}_0 \overset{tr_F^*}{\Longrightarrow}_{BC} \widehat{G}_n, G_T)$ with $G_S = \widehat{G}_n^S$, $\widehat{G}_n^T = G_T$ and $\widehat{G}_n^S = G_{n,0}$ for the corresponding source sequence $G_{00} \overset{tr_S^*}{\Longrightarrow} G_{n,0}$ (see Definition 26).

Proof. We have to show that in step 2 the forward consistent BC-match $(L_{n,F} \overset{d_n}{\hookleftarrow} D_n \overset{m_{n,F}}{\longrightarrow} \widehat{G}_{n-1})$ leads to a partial BC-match consistent step for $i = n$. In the following diagram the front faces are constructed as POs given by applying the forward consistent BC-match $(L_{n,F} \overset{d_n}{\hookleftarrow} D_n \overset{m_{n,F}}{\longrightarrow} \widehat{G}_{n-1})$. The left face is PB (1) from Definition 27 and the top face commutes by (2). Now the back face can be constructed as PO (3) leading to $G_{n-1,0} \overset{tr_{n,S}}{\Longrightarrow} G_{n,0}$. It remains to construct injective $\widehat{g}_{n-1} : G_{n,0} \to \widehat{G}_{n-1}^+$, s.t. bottom and right faces commute.



Since front left is PO with injective d'_n it is also PB s.t. the composition of left and front left is PB. The S-component of this composite PB is given by the following outer diagram, where all morphisms are injective, including the matches $m_{n,F}$ by assumption leading to injectivity of $m_{n,S}$. The PO (3) implies by Theorem for effective unions [Her08],[LS04] a unique f s.t. (4) and (5) commute and f is injective using that the outer diagram is an injective PB. Moreover, the PO-object $G_{n,0}$ in PO (3) can be chosen s.t. f becomes inclusion $G_{n,0} \subseteq \widehat{G}_{n-1}^{+S}$.



Now we define $\widehat{g}_{n-1} : G_{n,0} \to \widehat{G}^+_{n-1}$ by $\widehat{g}_{n-1,S} = f$ and $\emptyset$ for the C- and T-component this implies that bottom and right face in the cube above are commutative. Hence, we have also a partial BC-match consistent step for $i = n$ with inclusion $G_{n,0} \subseteq \widehat{G}^+_{n-1} \subseteq \widehat{G}_n$. □

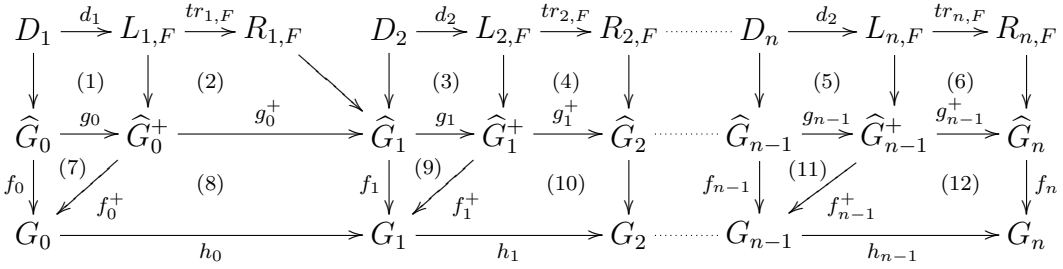
Remark
The on-the-fly construction can be executed y either depth-first search or breadth-first search. It halts, if no rule is further applicable and $G^S_{n,0} \neq \widehat{G}^S_n$. This means, that a transformation in this branch is not possible.

Now it would be interesting, in which way a forward transformation with borrowed context could be extended to a standard forward transformation without borrowed context. This is an essential Theorem of this thesis, because with such extension main parts of the theory of graph transformation with triple graph grammars can be adapted to our approach.

Theorem 5 (Extension of BC-Forward Transformations).
Given a BC-forward transformation $\widehat{G}_0 \overset{tr^*_F}{\Longrightarrow}_{BC} \widehat{G}_n$ with $\widehat{G}_0 = (\widehat{G}^S_0 \leftarrow \emptyset \rightarrow \emptyset)$ there is an extension to a forward transformation $G_0 \overset{tr^*_F}{\Longrightarrow} G_n$ with $G_0 = (\widehat{G}^S_n \leftarrow \emptyset \rightarrow \emptyset)$ and $G_n = \widehat{G}_n$.

Construction.

Given the BC-forward transformation $\widehat{G}_0 \overset{tr^*_F}{\Longrightarrow} \widehat{G}_n$ with $\widehat{G}^C_0 = \widehat{G}^T_0 = \emptyset$ by POs (1)-(6) and $G_0 = (\widehat{G}_{n,S} \leftarrow \emptyset \rightarrow \emptyset)$ we construct diagrams (7)-(12) in the S-, C-, and T-component as follows, where all morphisms are injective:



S-Component The S-components $g^+_{0,S}, g^+_{1,S}, ..., g^+_{n-1,S}$ are identical because of POs (2)-(6) and identical $tr^1_{F,S}, tr^2_{F,S}, ..., tr^n_{F,S}$. Then $h_{0,S}, h_{1,S}, ..., h_{n,S}$ are constructed as identities

in view of POs (8), (10) and (12):

$$\begin{aligned}
f_{1,S} &= g_{n-1,S} \circ ... \circ g_{1,S} : \widehat{G}_{1,S} \to \widehat{G}_{n,S} = G_{0,S} = G_{1,S} \\
&\vdots \\
f_{n,S} &= id : \widehat{G}_{n,S} \to \widehat{G}_{n,S} = G_{0,S} = G_{1,S} \\
f^+_{0,S} &= f_{1,S} : \widehat{G}^+_{0,S} = \widehat{G}_{1,S} \to G_{1,S} = G_{0,S} \\
f^+_{1,S} &= f_{2,S} : \widehat{G}^+_{1,S} = \widehat{G}_{2,S} \to G_{2,S} = G_{1,S} \\
&\vdots \\
f^+_{n-1,S} &= f_{n,S} : \widehat{G}^+_{n-1,S} = \widehat{G}_{n,S} \to G_{n,S} = G_{n-1,S}
\end{aligned}$$

C-Component The C-components $g_{0,C}, g_{1,C}, ..., g_{n-1,C}$ are identical because of POs (1), (3), (5) and identical $d_{1,C}, d_{2,C}, ..., d_{n,C}$. Then

$$\begin{aligned}
f^+_{0,C} = f_{0,C} = \emptyset \quad & \text{with } \widehat{G}_{0,C} = \widehat{G}^+_{0,C} = G_{0,C} = \emptyset \\
f_{1,C}, h_{0,C} \quad & \text{by PO (8) and } f^+_{1,C} = f_{1,C} \text{ using } g_{1,C} = id \\
f_{2,C}, h_{1,C} \quad & \text{by PO (10) and } f^+_{1,C} = f_{1,C} \text{ using } g_{2,C} = id \\
&\vdots \\
f_{n,C}, h_{n-1,C} \quad & \text{by PO (12)}
\end{aligned}$$

T-Component Replace C by T in *C-Component*.

Proof. First we show that diagrams (7)-(12) commute componentwise and (8), (10) and (12) are POs componentwise.

S-Component

$$\begin{aligned}
(7) \quad & f^+_{0,S} \circ g_{0,S} = f_{1,S} \circ g_{0,S} = f_{0,S} \\
(9) \quad & f^+_{1,S} \circ g_{1,S} = f_{2,S} \circ g_{1,S} = f_{1,S} \\
(11) \quad & f^+_{n-1,S} \circ g_{n-1,S} = f_{n,S} \circ g_{n-1,S} = id \circ g_{n-1,S} = f_{n-1,S}
\end{aligned}$$

(8), (10), and (12) are POs in the S-component because the horizontal morphisms are identities.

C-Component (7), (9), and (11) commute by construction, e.g. $f^+_{0,C} \circ g_{0,C} = f^+_{0,C} \circ id = f_{0,C}$.
(8), (10), and (12) are POs in the C-component by construction.

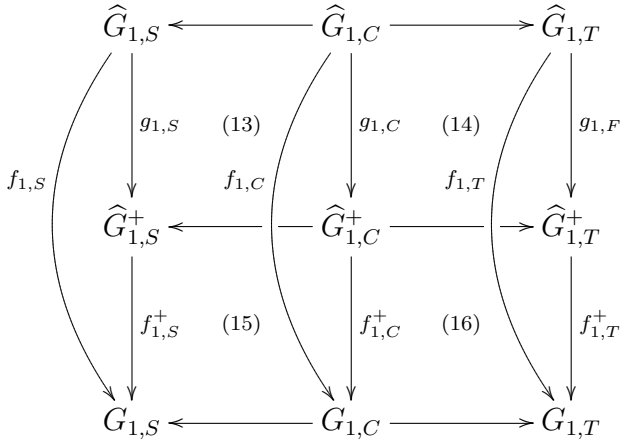
T-Component Similar to C-component.

Assume now that all new morphisms in (7)-(12) are TGG-morphisms such that (8), (10), and (12) become TGG-POs. In this case the forward transformation $G_0 \overset{tr_F^*}{\Longrightarrow} G_n$ is given by POs (2)+(8), (4)+(10), and (6)+(12) with $G_0 = (\widehat{G}_{n,S} \leftarrow \emptyset \rightarrow \emptyset)$ and $G_n = \widehat{G}_n$, because we have $f_n = id$, with $f_{n,S} = id$ by construction and $f^+_{0,C} = id$ implies by POs (8), (10), and (12) also $f_{n,C} = id$ and similar $f_{n,T} = id$.

It remains to show step by step that all morphisms in $(7) - (12)$ are TGG-morphisms and $G_1, ..., G_n$ are well-defines such that (8), (10), and (12) become TGG-POs.

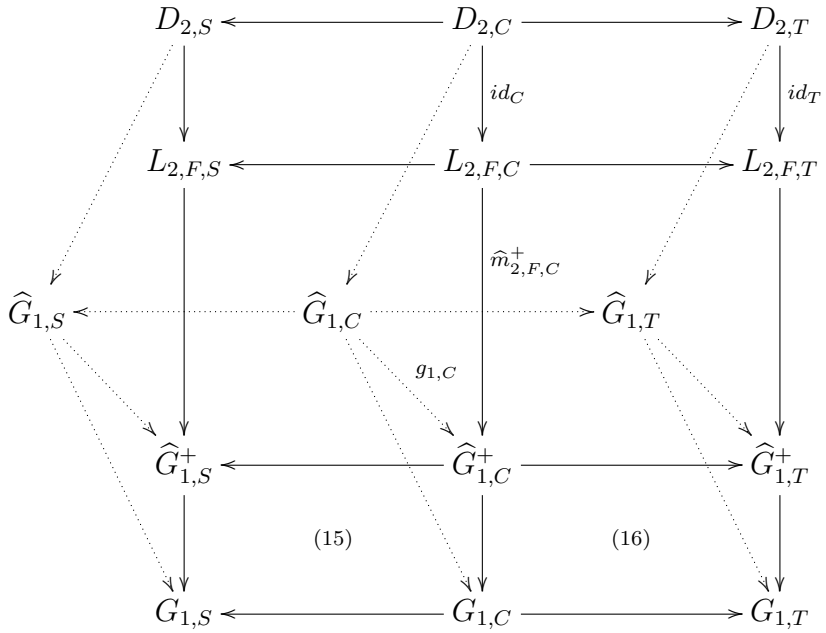
First of all $G_0 = (\widehat{G}_{n,S} \leftarrow \emptyset \rightarrow \emptyset)$ and $f_0 = (f_{0,S}, \emptyset, \emptyset)$, $f_0^+ = (f^+_{0,S}, \emptyset, \emptyset)$ are well-defined TGG-morphisms. Since (8) is already PO in each component there are unique graph morphisms $G_{i,C} \rightarrow G_{i,S}$ and $G_{i,C} \rightarrow G_{i,T}$ s.t. $G_1 = (G_{1,S} \leftarrow G_{1,C} \rightarrow G_{1,T})$ is a TGG-graph, f_1 and h_0 are TGG-morphisms, and (8) is a TGG-PO.

In order to show that f_1^+ is TGG-morphism we have to show that (15), (16) commute, while (13), (14) and the composite diagrams commute because g_1 and f_1 are TGG-morphisms respectively.



Since $\widehat{G}^+_{1,C}$ is PO in the C-component of (3) we have that $g_{1,C}$ and $\widehat{m}^+_{2,F,C}$ are jointly epi and it is sufficient to show that (15) and (16) commute if they are composed with these both morphisms. Concerning composition with $g_{1,C}$ this follows from commutativity of (13), (14), the vertical composed diagrams and the triangles in each component.

For the composition of (15) and (16) with $\widehat{m}^+_{2,F,C} = \widehat{m}^+_{2,F,C} \circ id_C$ we consider the following diagrams corresponding to diagram (3) and (9) in the construction, where id_C and id_T are identities by definition of BC-forward transformations.



We know by construction that all subdiagrams except for (15) and (16) commute. This implies that (15) and (16) composed with $\widehat{m}^+_{2,F,C}$ commute and hence also (15) and (16) commute using $(g_{1,C}, \widehat{m}^+_{2,F,C})$ are jointly epi. This implies that f_1^+ is TGG morphism.

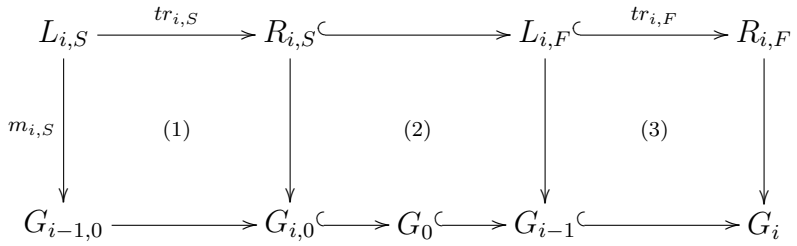
Similar to f_1 and h_0 above using TGG-morphism f_0^+ we can conclude now aunique TGG-graph G_2 s.t. (10) becomes TGG-PO with TGG-morphisms f_2 and h_1. Similar to f_1^+ also f_2^+ is TGG-morphism. This can be iterated for all $i = 0, ..., n$ s.t. the diagrams $(7) - (12)$ are TGG-diagrams with TGG-POs (8), (10), and (12).

□

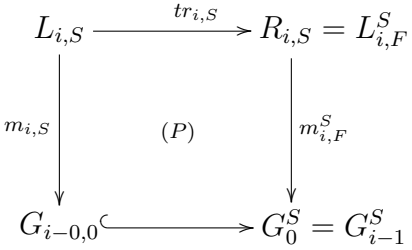
With this extension of a BC-forward transformation to a standard forward transformation, we need the notion of partial match consistency for this as well.

Theorem 6 (Extension of Partial BC-Match Consistency).
Given a partially BC-match consistens pair $(G_{00} \overset{tr^*_S}{\Longrightarrow} G_{n0}, \widehat{G}_0 \overset{tr^*_F}{\Longrightarrow}_{BC} \widehat{G}_n)$ with $G_{00} = \emptyset$, $G_{n0} \hookrightarrow \widehat{G}^+_{n-1} \hookrightarrow \widehat{G}_n$ then the extension construction leads to a *partially match consistent* sequence $G_{00} \overset{tr^*_S}{\Longrightarrow} G_{n0} \hookrightarrow G_0 \overset{tr^*_F}{\Longrightarrow} G_n$ in the sense of [EEHP09].

Remark
Partial match consistency of $G_{00} \overset{tr^*_S}{\Longrightarrow} G_{n0} \hookrightarrow G_0 \overset{tr^*_F}{\Longrightarrow} G_n$ means that we have for each $i = 1, ..., n$ the following diagram where (1) and (3) are POs, (2) commutes and (1) + (2) is PB.



Since (1) and (3) are POs by construction we essentially have to show that (1)+(2) is PB, which is equivalent that the S-component of (1) + (2) shown in diagram (P) is a PB.

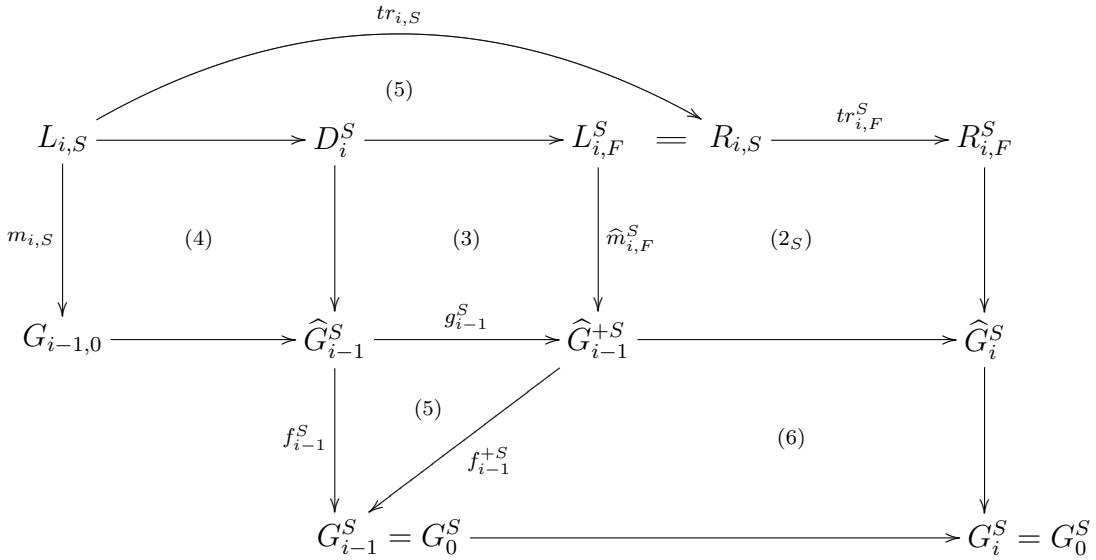


This means that we have to conclude from the diagram in definition 26 and the extension construction in theorem 5 that (P) is PB.

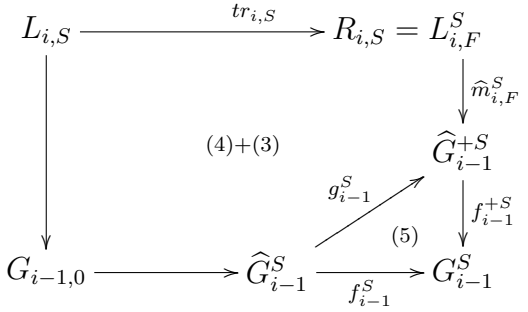
Proof. The partially BC-match consistent pair

$$(G_{00} \overset{tr^*_S}{\Longrightarrow} G_{no}, \widehat{G}_0 \overset{tr^*_F}{\Longrightarrow}_{BC} \widehat{G}_n)$$

with $G_{00} = \emptyset$, $G_{n0} \hookrightarrow \widehat{G}^+_{n-1} \hookrightarrow \widehat{G}_n$ laeds in the S-component to PBs (4) and (3) and PO (2_S) from PO (2), where (3) is PO and PB.



And from construction in Theorem 5 we have injective f_{i-1} and f^+_{i-1} leading in the S-component to commutative (5) and PO (6). Now diagram (P) from the remark of Theorem 6 corresponds to the following outer diagram:



Because $m^S_{i,F} = f^{+S}_{i-1} \circ \widehat{m}^S_{i,F}$ the inner part (4)+(3) is PB by partial BC-match consistency. Now f^{+S}_{i-1} mono implies that also the outer diagram (P) is PB (by extension property of PBs by monos). Finally we have an inclusion $G_{n0} \hookrightarrow \widehat{G}^+_{n-1} \hookrightarrow \widehat{G}_n$ leading to $G_{n0} \hookrightarrow G_0$ using $G_0 = (\widehat{G}^S_n \leftarrow \emptyset \rightarrow \emptyset)$ and hence to partial match consistency of $G_{00} \overset{tr^*_S}{\Longrightarrow} G_{n0} \hookrightarrow G_0 \overset{tr^*_F}{\Longrightarrow} G_n$.

□

In Theorem 5 we defined under which conditions a BC-forward transformation $\widehat{G}_0 \overset{tr^*_F}{\Longrightarrow}_{BC} \widehat{G}_n$ can be extended to a forward transformation $G_0 \overset{tr^*_F}{\Longrightarrow} G_n$. This leads to the Theorem 7, which allows us to extend BC-model transformation sequences to standard model transformation sequences.

Theorem 7 (Extension of BC-Model Transformation Sequences).
Each BC-model transformation $(G_S, \widehat{G}_0 \overset{tr^*_F}{\Longrightarrow}_{BC} \widehat{G}_n, G_T)$ with $G_S = \widehat{G}^S_n$, $G_T = \widehat{G}^T_n$ according to Theorem 4 can be extended by Theorem 5 to a model transformation sequence $(G_S, G_0 \overset{tr^*_F}{\Longrightarrow} G_n, G_T)$ with the same G_S, G_T satisfying $G_S = G^S_0$, $G_T = G^T_n$ and $G_n = \widehat{G}_n$.

Proof. By definition of the BC-model transformation sequence in Theorem 4 (part 3) we have for the BC-model transformation sequence given in our theorem a partial BC-match consistent pair $(G_{00} \overset{tr^*_S}{\Longrightarrow} G_{n0}, \widehat{G_0} \overset{tr^*_F}{\Longrightarrow}_{BC} \widehat{G}_n)$ with inclusion $G_{n,0} \hookrightarrow \widehat{G}_n$, $G_S = \widehat{G}^S_n$, $G_T = \widehat{G}^T_n$ and $\widehat{G^S_n} = G_{n,0}$. By Theorem 6 we have a partially match consistent sequence $G_{00} \overset{tr^*_S}{\Longrightarrow} G_{n0} \hookrightarrow G_0 \overset{tr^*_F}{\Longrightarrow} G_n$. By Theorem 5 we have $G_n = \widehat{G}_n$ and $G^S_0 = \widehat{G}^S_n$. Hence we have $G_{n0} = \widehat{G}^S_n = G^S_0$, which means that according to Theorem 1 in [EEHP09] a model transformation sequence $(G'_S, G_0 \overset{tr^*_F}{\Longrightarrow} G_n, G'_T)$ with $G'_S = G^S_0$ and $G'_T = G^T_n$. Finally we have $G'_S = G^S_0 = \widehat{G}^S_n = G_S$ and $G'_T = G^T_n = \widehat{G}^T_n = G_T$ by Theorem 4. Hence we have model transformation $(G_S, G_0 \overset{tr^*_F}{\Longrightarrow} G_n, G_T)$ with $G_S = G^S_0$ and $G_T = G^T_n$. □

3.3 Correctness, Completeness and Termination

It essential to determine when a model transformation generates models that are correct, complete and whether a transformation terminates. In Section 2.2 the Theorem 3 for correctness, completeness and termination of a standard model transformation was given. In this Section we will now define this for the BC-model transformation.

Definition 29 (Correctness).
A BC-model transformation is *correct*, if for all BC-model transformation sequences $(G_S, \widehat{G}_0 \overset{tr_F^*}{\Longrightarrow}_{BC} \widehat{G}_n, G_T)$ we have $\widehat{G}_n \in VL$ with $\widehat{G}_0^S \subseteq \widehat{G}_n^S = G_S \in VL_S$ and $\widehat{G}_n^T = G_T \in VL_T$.

Theorem 8
Each BC-model transformation is correct.

Proof. From Theorem 7 follows, that a model transformation with borrowed context $(G_S, \widehat{G}_0 \overset{tr_F^*}{\Longrightarrow}_{BC} \widehat{G}_n, G_T)$, where $G_S = \widehat{G}_n^S$, $G_T = \widehat{G}_n^T$, and $\widehat{G}_0^S \subseteq \widehat{G}_n^S$, can be extended to a model transformation sequence $(G_S, G_0 \overset{tr_F^*}{\Longrightarrow} G_n, G_T)$, where $G_S = G_0^S$, $G_T = G_n^T$, and $G_n = \widehat{G}_n$. Additionally, from Theorem 3 follows, that this transformation sequence is correct, i.e., $G_n \in VL$ with $G_S \in VL_S$ and $G_n \in VL_T$, using $G_n = \widehat{G}_n$ this implies $\widehat{G}_0^S \subseteq \widehat{G}_n^S = G_S \in VL_S$ and $\widehat{G}_n^T = G_T \in VL_T$. □

Definition 30 (Termination).
A model transformation with borrowed context based on forward translation rules *terminates*, if each BC-source consistent sequence can not be extended further by any transformation step.

Theorem 9 (Termination of BC-model transformations).
The on-the-fly-construction (according to Theorem 4) at $\widehat{G}_0 = (G_S \leftarrow \emptyset \rightarrow \emptyset)$ terminates for all source models G_S typed over TG^S, i.e. only finitely many steps are performed, if the following conditions are satisfied:

1. all source rules are creating
2. all matches $m_{i,F}$ are required to satisfy $D_i \neq L_{i,S}$ (see below).
3. G_S is finite, TR is finite and the rule component L and R of each triple rule $tr \in TR$ are finite.

Proof. Assume the procedure does not terminate for a finite G^S typed over TG, i.e. the derivation tree is infinite. Then we have

case 1: The procedure generates a partial BC-match consistent pair $(\emptyset \overset{tr_S^*}{\Longrightarrow} G_{n,0}, \widehat{G_0} \overset{tr_F^*}{\Longrightarrow}_{BC}$

$\widehat{G}_n$) with $n > |G^S| = |\widehat{G}_n^S|$, where $|G^S|$ denotes the amount of nodes and edges in G^S. From this follows:

a) in each step k elements are *borrowed* (nodes and edges): $k = |\widehat{G}_{i-1}^* \setminus \widehat{G}_{i-1}| = |L_{i,F} \setminus D_i| = |R_i^S \setminus D_i^S|$ and $D_i^S \subseteq R_i^S$

b) l elements are *created* in the source component in a step $G_{i-1,0} \overset{tr_{i,S}}{\Longrightarrow} G_{i,0}$: $l = |G_{i,0} \setminus G_{i-1,0}| = |R_{i,S} \setminus L_{i,S}| = |R_i^S \setminus L_i^S|$ and $L_i^S \subseteq D_i^S$, $L_i^S \neq D_i^S$

From a) and b) follows, that $l > k$, i.e. in each step there are more elements created than borrowed. So after n steps ($G_{00} \overset{tr_S^*}{\Longrightarrow} G_{n0}, \widehat{G}_0 \overset{tr_F^*}{\Longrightarrow}_{BC} \widehat{G}_n$) we have that $|G_{n0}^S| > |G^S| = |\widehat{G}_n^S|$, because each source step created more element than the corresponding forward step with BC. Hence, $\widehat{g}_{n+1}^S$ is not injective and therefore $\widehat{g}_{n+1}$ is not injective.

case 2: There are infinitely many steps $\widehat{G}_{n-1} \overset{tr_{n,F}}{\Longrightarrow}_{BC} G_n$ with forward consistent BC-matches, but each sequence is of finite length. Hence, we have infinitely many sequences starting at $\widehat{G}_0$ and because the derivation tree is inifinite we have infinitely many matches at one graph $\widehat{G}_i$.

We have a conflict because of the finiteness of TR. Furthermore, due to the fact that for each $tr_F \in TR_F$ there exist finitely many matches, because $\widehat{G}_i$ is finite, which is because $\widehat{G}_0$ is finite and rule compositions are finite.

□

4 Propagation of Graph Constraints Using Borrowed Context

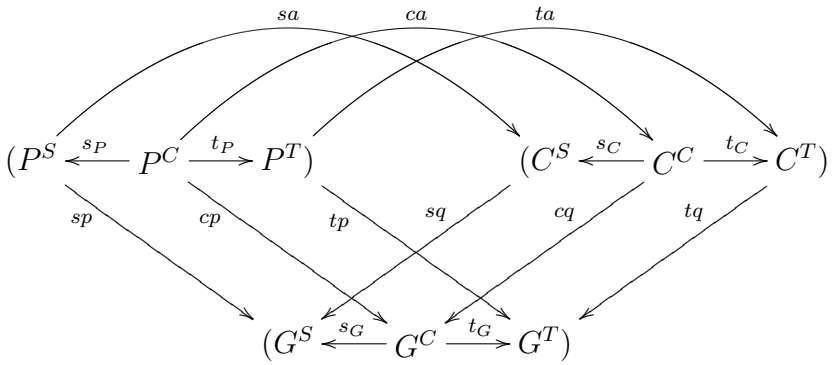
In this section another main result of this diploma thesis will be introduced. In Section *4.1 Graph Constraints* we will see, how we can formulate properties for graphs. How we can propagate such a constraint using the graph transformation with borrowed context will be shown in Section *4.2 Model Transformation of Graph Constraints*. Lastly in Section *4.3 Preservation of Model Properties* we will show, under which conditions a model satisfies a propagated graph constraint.

4.1 Graph Constraints

The notion of graph constraints for plain graphs was already introduced in Definition 6 in Section 2.1. For triple graphs we need to lift this definition to triple graph constraints.

Definition 31 (triple graph constraints).
An *atomic triple graph constraint* is of the form $PC(a)$ and consists of triple graphs P and C, and a triple graph morphism $a = (sa, ca, ta) : P \to C$.

A *triple graph constraint* is a Boolean formula over atomic triple graph constraints. This means that true and every atomic triple graph constraint are graph constraints, and, for triple graph constraints c and c_i with $i \in I$ for some index set I, $\neg c$, $\wedge_{i \in I} c_i$, and $\vee_{i \in I} c_i$ are graph constraints:



A triple graph $G : (G^S \leftarrow G^C \rightarrow G^T)$ *satisfies* a triple graph constraint c, written $G \models c$, if

- $c = true$;
- $c = PC(a)$ and, for every injective triple graph morphism $p : P \rightarrow G$, there exists an injective triple graph morphism $q : C \rightarrow G$ such that $q \circ a = p$;
- $c = \neg c'$ and G does not satisfy c';
- $c = \wedge_{i \in I} c_i$ and G satisfies all c_i with $i \in I$;
- $c = \vee_{i \in I} c_i$ and G satisfies some c_i with $i \in I$;

Using the notion of triple graph constraints we can now formulate conditions and properties on triple graphs, i.e. on

4.2 BC-Model Transformation of Graph Constraints

When we want to transform a graph constraint, we not just have to transform a single model fragment, but two fragments P and C with the morphism $a : P \rightarrow C$ simulatneously. Moreover, we have to confirm, if the parts we borrow in P are already in C and hence needs not to be borrowed or if C misses parts of them as well, so it need to be borrowed in the same way. Additionally, the morphism a needs to be translated such that it maps the borrowed parts as well.

Definition 32 (Propagation of a Graph Constraint).
Given a graph constraint $PC(a) : P \xrightarrow{a} C$ for the source language of a model transformation

with borrowed context MT_{BC}. A propagated integrated constraint $PC(\widehat{a}) : \widehat{P} \xrightarrow{\widehat{a}} \widehat{C}$ of $PC(a)$ satisfies the following conditions.

1. There is a BC-source consistent forward sequence

$$(S1): \quad ((P \leftarrow \emptyset \rightarrow \emptyset) \overset{tr_F^*}{\Longrightarrow}_{BC} \widehat{P}_l)$$

with $\widehat{P}_l = \widehat{P}$.

2. There is a BC-forward transformation sequence

$$(S2): \quad ((C \leftarrow \emptyset \rightarrow \emptyset) \overset{tr_F^*}{\Longrightarrow}_{BC} \widehat{C}_l \overset{tr_F'^*}{\Longrightarrow}_{BC} \widehat{C}_n)$$

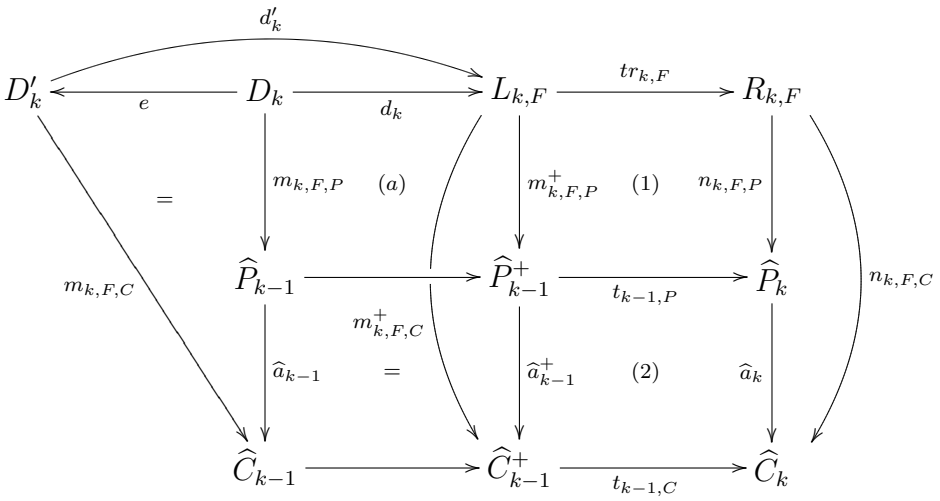
with $\widehat{C}_n = \widehat{C}$ s.t.

3. $(S2)$ is an extension of $(S1)$ as shown in the diagram below and the triple graph morphism $\widehat{a} : \widehat{P} \rightarrow \widehat{C}$ is given by $\widehat{a} = trace(tr_F'^* \circ \widehat{a}_0)$.

$$\begin{array}{ccccccc}
(P \leftarrow \emptyset \rightarrow \emptyset) & = & \widehat{P}_0 & \overset{tr^*}{\Longrightarrow} & \widehat{P}_l & = & \widehat{P} \\
 & & \downarrow{\scriptstyle (a,\emptyset,\emptyset)} & & \downarrow{\scriptstyle \widehat{a}_0} & & \searrow{\scriptstyle \widehat{a}} \\
(C \leftarrow \emptyset \rightarrow \emptyset) & = & \widehat{C}_0 & \overset{tr^*}{\Longrightarrow} & \widehat{C}_l & \overset{tr_F'^*}{\Longrightarrow} \widehat{C}_n = & \widehat{C}
\end{array}$$

Moreover, for each propagated integrated constraint $PC(\widehat{a})$ we define the propagated source constraint by $PC(\widehat{a}^S) : \widehat{P}^S \xrightarrow{\widehat{a}^S} \widehat{C}^S$ and the propagated target constraint by $PC(\widehat{a}^T) : \widehat{P}^T \xrightarrow{\widehat{a}^T} \widehat{C}^T$.

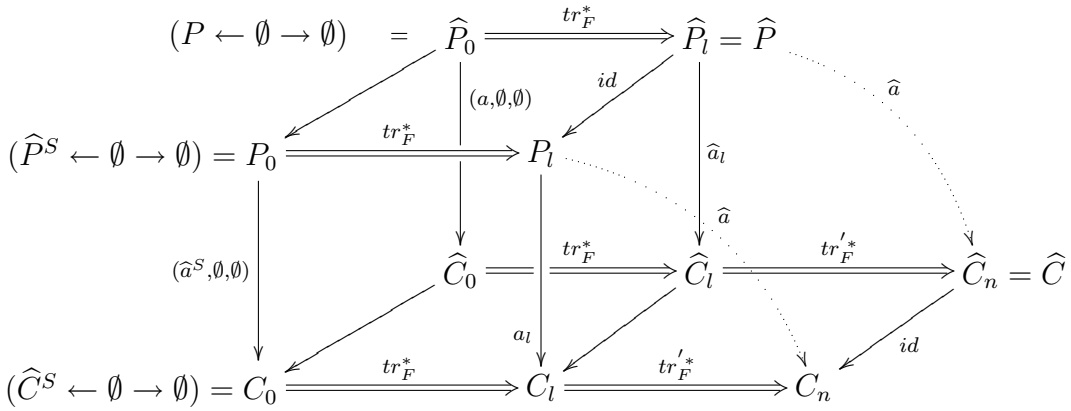
Remark
$(S2)$ extension of $(S1)$ means $(S2)$ is a BC-forward transformation sequence via $tr_F^*, tr_F'^*$, where the matches $m^+_{k,F,P} : L_{k,F} \rightarrow \widehat{P}_k$ of sequence $(S1)$ are extended in the first l steps to matches $m^+_{k,F,C} : L_{k,F} \rightarrow \widehat{C}_k$ with $m^+_{k,F,C} = \widehat{a}^+_{k-1} \circ m^+_{k,F,P}$ and $\widehat{a}^+_{k-1}$ induced by PO (a) as shown in the diagram below. Moreover $\widehat{a}_k$ is induced by PO (1) and (2) becomes PO.



In the first step we have $m^+_{1,F,C} = (a, \emptyset, \emptyset) \circ m_{1,F,P}$. At each step we require an $e : D_k \to D'_k$ with $d'_k \circ e = d_k$ and $m_{k,F,C} \circ e = \widehat{a}_{k-1} \circ m_{k,F,P}$.

Remark
The propagation of graph constraints can be performed using the on-the-fly construction for BC-model transformations according to Thm. 4, where termination can be ensured according to Thm. 9. For a single graph constraint $PC(a) : P \xrightarrow{a} C$ there may be several propagated constraints $PC(\widehat{a}) : \widehat{P} \xrightarrow{\widehat{a}} \widehat{C}$ that differ in the derived premise $\widehat{P}$ and furthermore, they can differ in the derived conclusion $\widehat{C}$ for a single derived $\widehat{P}$. Moreover, there may not exist a single propagated constraint for a graph constraint $PC(a) : P \xrightarrow{a} C$.

Remark
The propagation of a graph constraint can be extended to the following commutative diagram, where we replace the transformation sequences tr^*_F, $tr^{*'}_F$ by the corresponding traces $trace(tr^*_F)$, $trace(tr^{*'}_F)$.



Example 5 (Propagation of a Graph Constraint).
A simple graph constraint is given in Figure 7. The mapping of the morphism a is indicated by the labelling of the nodes.

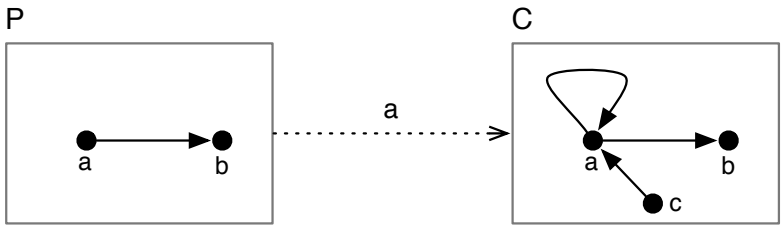


Figure 7: A graph constraint

Now assume, we want to transform the constraint according to Definition 32 with the forward rule shown Figure 8 as the graphs L_1 and R_1. We find a partial match $m^-_{p1} : L_1 \rightharpoonup$

$\widehat{P}_0$ leading to the span $(\widehat{P}_0 \leftarrow D_{p1} \rightarrow L_1)$ and a pushout object $\widehat{P}_0^+$. Additionally, there exists a partial match $m_{c1}^- : L_1 \rightharpoonup \widehat{C}_0$ (not in the figure), which leads to the span $(\widehat{C}_0 \leftarrow D_{c1} \rightarrow L_1)$ and a pushout object $\widehat{C}_0^+$. Moreover, there exists a morphism $e : D_{p1} \rightarrow D_{c1}$, such that the triangle commutes. Therefore, the morphism $\widehat{a}_0^+$ is induced as shown in the Figure. Over the pushouts in the transformation step the morphism $\widehat{a}_1$ is induced.

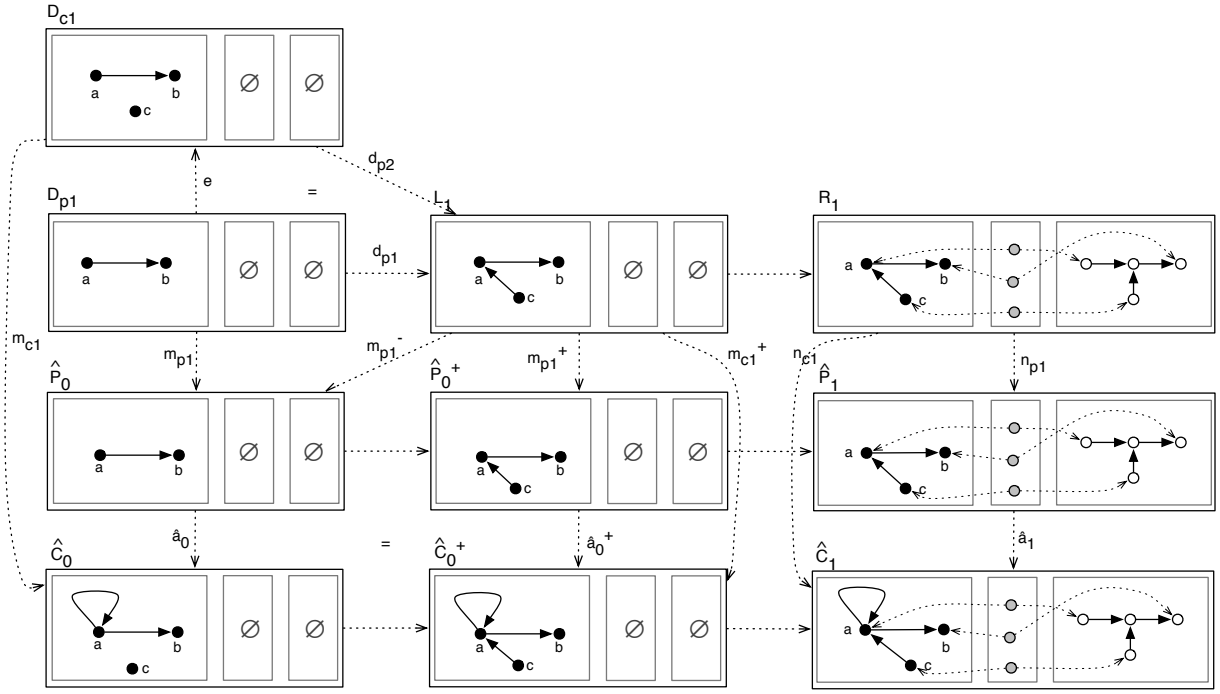


Figure 8: First Step of a Propagation of a Graph Constraint

Now, the transformation of the premise is completed, but the edge in the conclusion at the node a needs still to be translated. Therefore, we apply a borrowed context transformation to the triple graph $\widehat{C}_1$ which is shown in Figure 9.

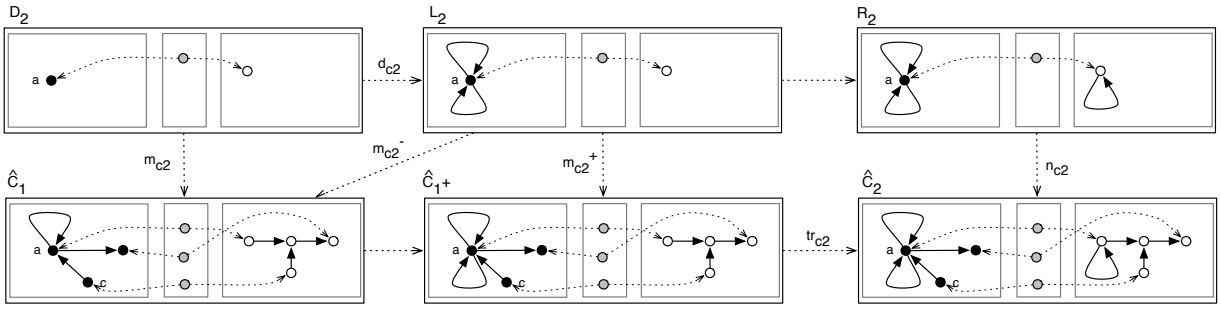


Figure 9: Second Step of a Propagation of a Graph Constraint

After we transformed both model fragments of the graph constraint completely, the

result is the propagated triple graph constraint shown in Figure 10, with $\widehat{P} = \widehat{P}_1$, $\widehat{C} = \widehat{C}_2$, and $\widehat{a} = tr_{c2} \circ \widehat{a}_1$. In Theorem 10 we will show for each integrated model G that $G^S \models PC(\widehat{a}^S)$ implies $G \models PC(\widehat{a})$,. This means in our example, if we require in the source component G^S of a model G the two self loops at the node a in the given pattern, we require one self loop at the corresponding node in the target component of the model G with the pattern given.

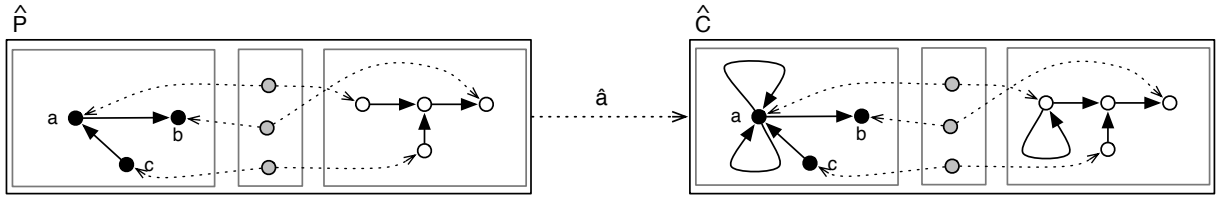


Figure 10: Propagated Constraint

In the following, we define some general properties of model transformations without concerning borrowed context. These properties will be used to show the main result of this chapter about the preservation of model properties by the propagation of graph constraints in Thm. 10 in Sec. 4.3.

Definition 33 (Guaranteed Completion).
A model transformation $MT : VL_{S0} \Rrightarrow VL_{T0}$ based on forward rules *guarantees completion*, if each partially source consistent forward sequence $G_0 \overset{tr_F^*}{\Longrightarrow} G_k$ can be extended to a source consistent forward transformation $G_0 \overset{tr_F^*}{\Longrightarrow} G_k \overset{tr_F'^*}{\Longrightarrow} G_n$.

Remark
A model transformation with guaranteed completion can be executed using the on-the-fly construction [EEHP09] and the condition ensures that no backtracking is necessary.

Definition 34 (Source and Target Uniqueness of Morphisms).
A triple graph G ensures *source uniqueness of morphisms*, if for any two injective triple morphism $p : P \to G$ and $p' : P \to G$ we have that $p^S = p'^S \Rightarrow p = p'$. *Target uniqueness of morphisms* is defined analogously by replacing "S" with "T". A language VL of integrated models ensures source (resp. target) uniqueness of morphisms, if each triple graph $G \in VL$ ensures source (resp. target) uniqueness of morphisms.

Definition 35 (Strong Functional Behaviour).
A model transformation based on forward rules has *strong functional behaviour*, if for each source model $G_S \in VL_S$ and any two model transformation sequences $(G_S, G_0 \overset{tr_F^*}{\Longrightarrow}$

$G_n, G_T)$ and $(G_S, G_0 \overset{tr'^*_F}{\Longrightarrow} G'_m, G'_T)$ we have that $G_n \cong G'_m$ and $G_T \cong G'_T$. Moreover, the transformation sequences $G_0 \overset{tr'^*_F}{\Longrightarrow} G_n$ and $G_0 \overset{tr^*_F}{\Longrightarrow} G'_m$ are switch-equivalent.

4.3 Preservation of Model Properties

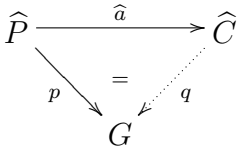
In this section we define the conditions under which a propagated graph constraint is a constraint in the target language, i.e., under which conditions a model from the target language satisfies the propagated constraint. We can show, that we can determine, when an integrated model satisfies an propagated graph constraint, that is an integrated model as well.

Theorem 10 (Translated Graph Constraint).
Given a graph constraint $PC(a) : P \xrightarrow{a} C$and a model transformation MT based on forward rules with strong functional behaviour, guaranteed completion and source uniqueness of morphisms for the triple language VL. Let $PC(\widehat{a})$ be a propagated integrated constraint of $PC(a)$ with $PC(\widehat{a}^S)$ and $PC(\widehat{a}^T)$ propagated source resp. target constraint. Then, an integrated model $G = (G^S \leftarrow G^C \rightarrow G^T) \in VL$ satisfies $PC(\widehat{a})$ and G^T weakly satisfies $PC(\widehat{a}^T)$, if the source model G^S satisfies the source constraint $PC(\widehat{a}^S)$, i.e.

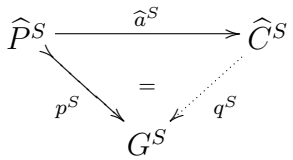
$$\begin{array}{lll} \text{if} & G^S & \models PC(\widehat{a}^S) \\ \text{then} & G & \models PC(\widehat{a}) \text{ and } G^T \models^w PC(\widehat{a}^T), \end{array}$$

where weak satisfaction $G^T \models^w PC(\widehat{a}^T)$ means for all injective $p^T : \widehat{P^T} \rightarrow G^T$ which can be extended to injective integrated morphism $p : \widehat{P} \rightarrow G$ exists injective $q^T : \widehat{C}^T \rightarrow G^T$ with $q^T \circ \widehat{a}^T = p^T$.

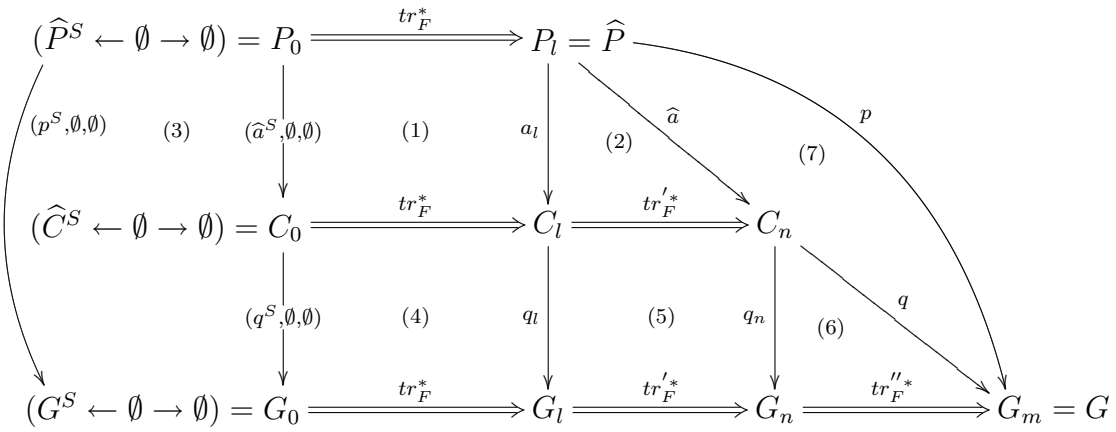
Proof. Given $p : \widehat{P} \rightarrowtail G$ we have to find $q : \widehat{C} \rightarrowtail G$ with



By assumption we have $q^S : \widehat{C}^S \rightarrowtail G^S$ with



By Remark (Extension of Propagation) we have commutative (1) and (2) where transformation sequences are replaced by their traces.



(3) commutes by assumption above and tr_F^* can be applied with match $(p^S, \emptyset, \emptyset) : P_0 \to G_0$ leading step by step to $G_0 \stackrel{tr_F^*}{\Longrightarrow}$ with (4) commuting and furthermore to $G_l \stackrel{tr_F'^*}{\Longrightarrow} G_n \stackrel{tr_F''^*}{\Longrightarrow} G_m = G$ by guaranteed completion with (5) commuting.

Let injective $q : \widehat{C} \to G$ be defined by $q = trace(tr_F''^*) \circ q_n$ s.t. (6) commutes. By assumption on G we have $\emptyset \stackrel{tr'''^*_F}{\Longrightarrow} G$ and by decomposition then a model transformation sequence $(G^S, G_0 \stackrel{tr'''^*_F}{\Longrightarrow} G, G^T)$ where $G_0 \stackrel{tr'''^*_F}{\Longrightarrow} G$ and $G_0 \stackrel{tr_F^*}{\Longrightarrow} G_l \stackrel{tr_F'^*}{\Longrightarrow} G_n \stackrel{tr_F''^*}{\Longrightarrow} G_m = G$ are equal up to switch equivalence by assumption on strong functional behaviour. This means that the corresponding traces are equal.

Note that commutativity of (1) and (2) implies that $\widehat{a}^S$ in (1) is the S-component of $\widehat{a}$ in (2) and that of (4) − (6) that q^S in (4) is the S-component of q in (6). It remains to show $q \circ \widehat{a} = p$ in (7). Due to source uniqueness of morphisms $q^S \circ \widehat{a}^S = p^S$, which is valid by assumption, implies $q \circ \widehat{a} = p$.

Finally, given $p^T : P^T \rightarrowtail G^T$ let $p : \widehat{P} \rightarrowtail G$ be extension of p^T then $G \models PC(\widehat{a})$ implies $q : \widehat{C} \to G$ with $g \circ \widehat{a} = p$. This implies $q^T \circ \widehat{a}^T = p^T$.

□

Remark
By Thm. 10 above we have shown that under the given side conditions all constraints that are valid for a source model $G^S \in VL_S$ are preserved and valid as integrated constraints for an integrated model $G \in VL$ containing G^S and an target construct for the target model G^T. Intuitively, this means that properties of the source model are preserved during the transformation and become valid for the integrated model and the target model.

Remark

1. According to Thm. 3 [HEOG10] functional behaviour of a model transformation based on forward rules can be analyzed using critical pair analysis of the tool AGG [AGG] with the derived forward translation rules, which extend the forward rules by additional boolean valued translation attributes. As shown in [HEGO10], the absence of significant critical pairs for the system of forward translation rules ensures

strong functional behaviour, which implies that there is no need for backtracking of the on-the-fly construction, which implies guaranteed completion.

2. Furthermore, strong functional behaviour ensures that each source consistent forward sequence is unique up to switch equivalence and therefore, the resulting triple graphs are unique up to isomorphism.

3. If no significant critical pairs are present we can furthermore conclude that each substructure of a graph $G^S \in VL_S$, which is translated by a sequence of forward steps, can only be translated by a switch-equivalent sequence of these forward steps. The reason is that critical pairs are complete (see [EEPT06]), i.e. for each parallel dependent pair of transformation steps there is a critical pair. Therefore, each substructure is uniquely translated implying source uniqueness of morphisms for the triple language VL.

For this reason, we can check guaranteed completion (Def. 35), strong functional behaviour (Def. 35) and source uniqueness of the triple language VL (Def.34) by checking the absence of critical pairs for the system of forward translation rules using the tool AGG.

5 Case Study

In this section to case studies will be introduced which will show examples for the model transformation of model fragments. The visual language ABT-Reo for Enterprise modelling will be introduced in section *5.1 Enterprise Modelling with ABT-Reo*. We will first show a standard model transformation in Section *5.2 Model Transformation of Models in ABT-Reo*, before showing in Section *5.3 Model Transformation of Constraints in ABT-Reo* how to transform a graph constraint of this language using borrowed context. Finally, in section *5.4 Model Transformation of Operational Semantics: From State Machines to Petri Nets* we present a second case study on transforming operational semantics between State Machines and Petri Nets.

5.1 Enterprise Modelling with ABT-Reo

Enterprise modeling encompasses the development of business and IT models of different domain-specific modeling languages. For the organization Credit Suisse a new framework was invented to connect both languages with triple graphs [Sch94]. This framework allows on the one hand for intra-modelling, using graph grammars and graph constraints, and on the other hand it allows for inter-modelling, with triple graph grammars used for model transformation and integration, as well.

One main part of the modelling framework is the use of the ABT & Reo for IT & Business service models, where ABT stands for abstract behaviour types and Reo stands for Reo connectors. The connection between the business universe and the IT universe is specified by a triple graph grammar (TGG).

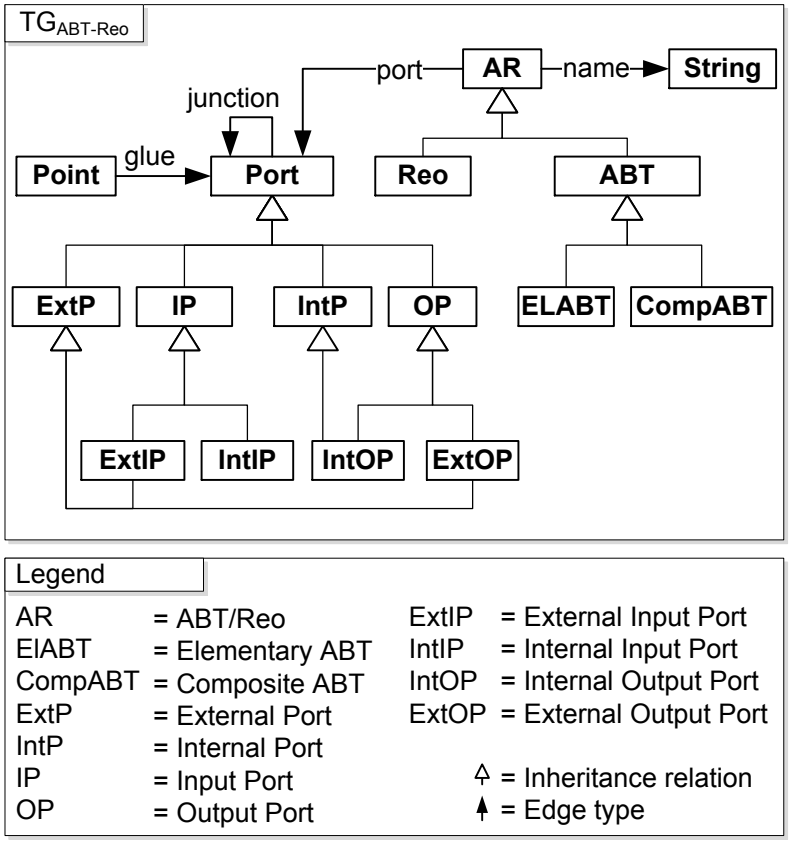


Figure 11: Type Graph $TG_{ABT-Reo}$ for ABT-Reo models

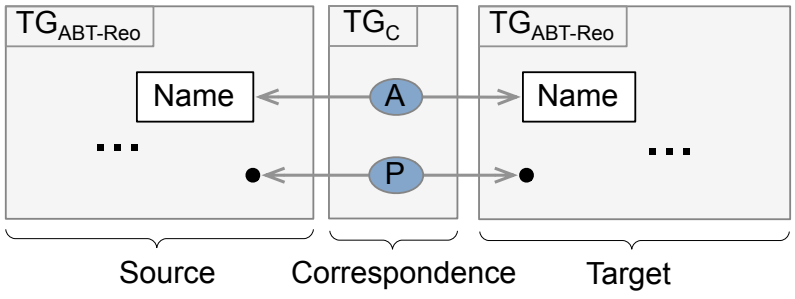


Figure 12: Triple Type Graph $TG_{ABT-Reo}$ for the Triple Graph Grammar

Since requirements for ABT-Reo models are specified by graph constraints, it is well suited as a case study for a model transformation with the borrowed context approach.

A triple graph $G = (G^S \xleftarrow{s_G} G^C \xrightarrow{t_G} G^T)$ in ABT-Reo consists of typed source and target graphs G^S and G^T, where the business service models are in the source component and the IT service models are in the target component and the graph G^C defines the correspondences. Both model views are ABT-Reo diagrams typed over the same type graph (see Fig. 11) that specifies the general structure of the graphs. The type graph for the integrated models in the triple graphs is shown in Figure 12.

Graphs typed over this graph are titled as ABT-Reo diagrams in abstract syntax. The type graph contains the main types "ABT" for abstract behaviour type nodes, "Reo" for Reo connectors, "Port" for ports and "Point" for points that glue together input and output ports of ABT nodes and Reo connectors. Additionally, there are ports that are used for external communication with other elements, i.e., other ABT nodes and Reo connectors. But ABT nodes can also be composite, i.e. they contain a further specified internal structure involving other ABT nodes and Reo connectors. The external ports of such composite nodes are connected to complementery internal ports, such that the communication is transferred through the borders of the composite ABT nodes.

Models based on such a graph in abstract syntax with the information of this graph encoded in the shapes of the models are titled as ABT-Reo diagrams in concrete syntax.

In Figure 13, there is an example for a model of the IT Universe in concrete syntax specifying the structure of a part of a network composed of local area networks (LANs). It consists of four ABT elements, with the outer ABT elements representing the LANs "NW4" and "NW7" and the two inner ones representing encoding/decoding nodes, i.e.the communication between both LANs on public network connections is encoded. The edges between the ABT elements are representations for the Reo connectors.

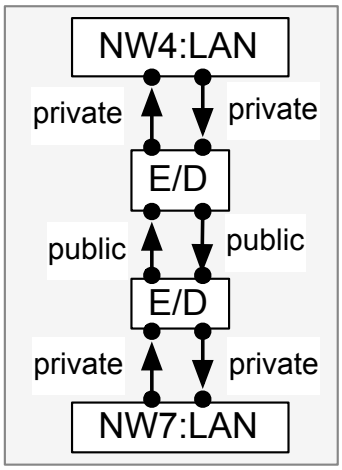


Figure 13: Example for an ABT-Reo model in concrete syntax of the IT Universe

To explain the relation between models in concrete and abstract syntax in further detail,

we can see in Figure 14 a fragment of the model in Figure 13 with the upper two ABT elements and their Reo connectors on the left side and the same part in abstract syntax on the right side.

The boxes in the left model correspond to the "ABT" nodes in the right, which have an edge to a node containing their types as a string, e.g the upper ABT element corresponds with the upper ABT node. The arrows on the left side correspond to the nodes of the type "Reo" on the right side, e.g. the arrow on the left in the left model corresponds to the Reo node on the left in the right model. The external input resp. output ports of the ABT and Reo elements are represented as nodes as well in the abstract syntax, but are not visualized in the concrete syntax. However, the left model contains bold bullets, gluing the arrows and the boxes together, representing the nodes of the type "Point", which connect an output port of an ABT element with the input port of a Reo connector and vice versa, e.g. the communication data exits an ABT element through its external output port and enters a Reo connector through the external input port, which is glued to it.

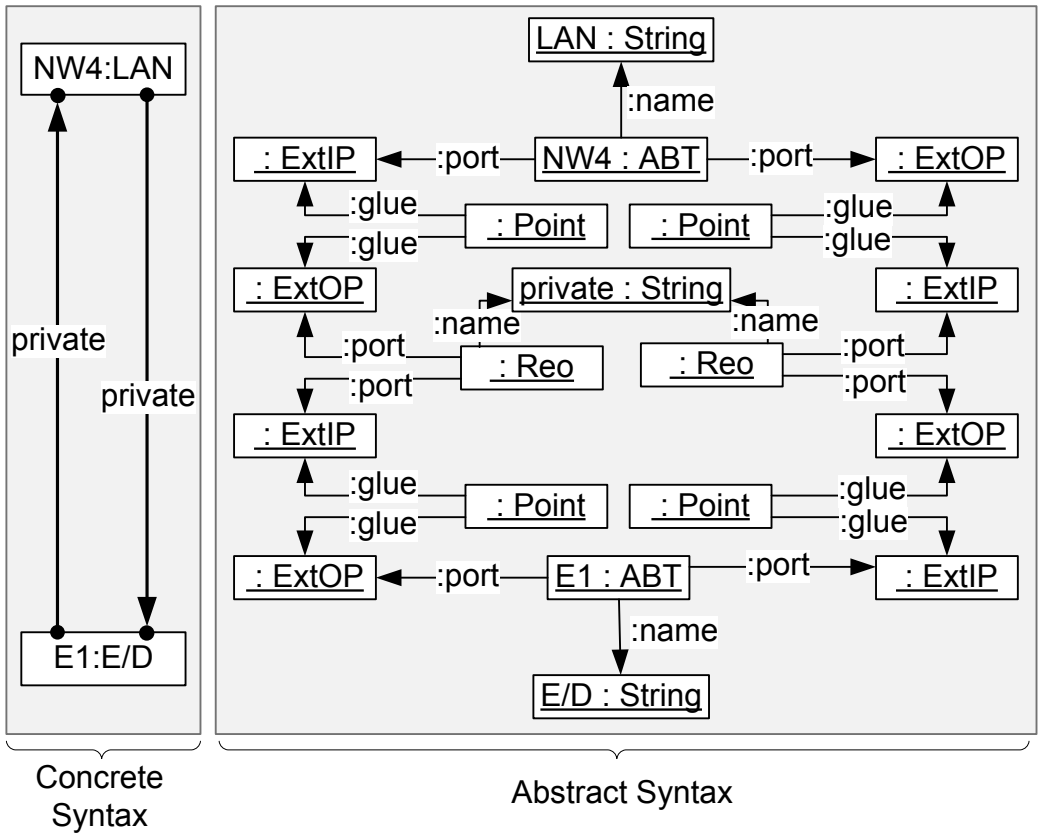


Figure 14: A part of the model in figure 13 in concrete and abstract syntax

As the direct comparison of concrete and abstract syntax shows, the concrete syntax is more compact. Therefore it is more intuitive, while the abstract syntax allows for a precise and detailed specification and analysis.

We will genrally work with the concrete syntax in this case study, but it is important to keep in mind, that it is just a curtailed version of the abstract syntax.

ABT-Reo has been modified to a triple graph grammar by [BHE09], where the models between the IT and business domain are connected via a correspondence graph and the one model as the source graph and the other as the target.

We first want to introduce the transformation rules. Because we work with a triple graph grammar, we have only non-deleting rules, that generate models and we only have a triple graph morphism from the left-hand-side (LHS) L of a rule to the right-hand-side (RHS) R without a graph K lying in between. As we can see in figure 15, in the complete notation, we have the representation of both sides. Whereas in the compact notation we use green coloring and the sign of a double plus sign to indicate, which elements are created by the rule, i.e. which elements are new on the RHS. In this study we will use rules in the compact notation in the concrete syntax.

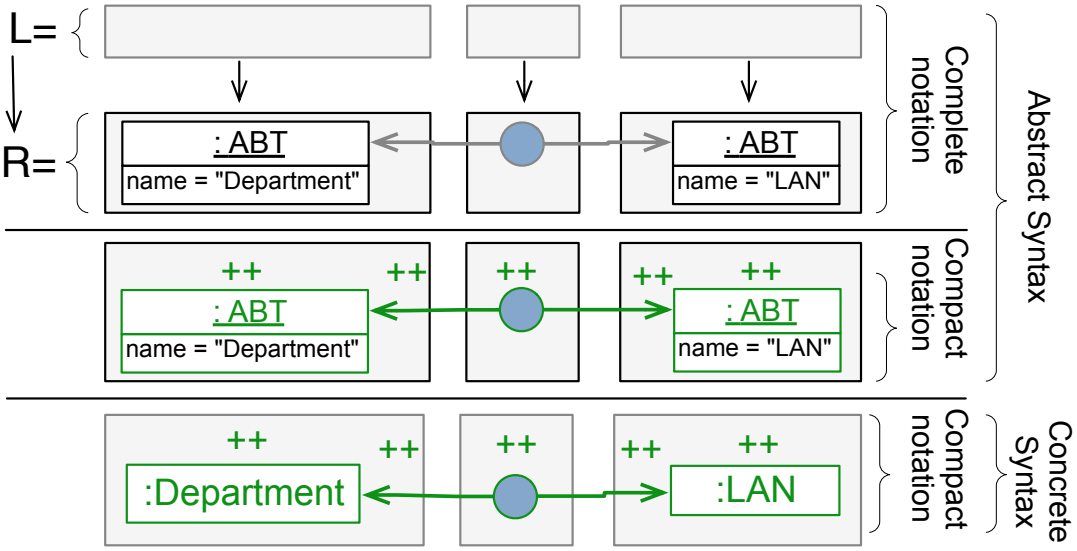


Figure 15: Different notations of the Triple Rule 0 - LAN2Department

For the transformation of the graph constraint later in this section we first need rules. The first transformation rule is given by figure 16. It is a forward rule, which has nothing in the LHS, shown by the fact, that there are only green elements with a double plus sign in the compact notation. The generated elements are similar in both models. A Reo connector with the name "public" with a gluing element at the beginning and at the end. The correspondence graph connects both element through these gluing elements.

The transformation rule shown in Figure 17 does not have an empty LHS. The parts, that are needed for this rule to be applicable, are the elements, generated by the rule *public2public*, i.e. the public Reo connectors and its gluing elements, which are corresponding with each other. The elements added then by the RHS are the encoding/decoding ABT

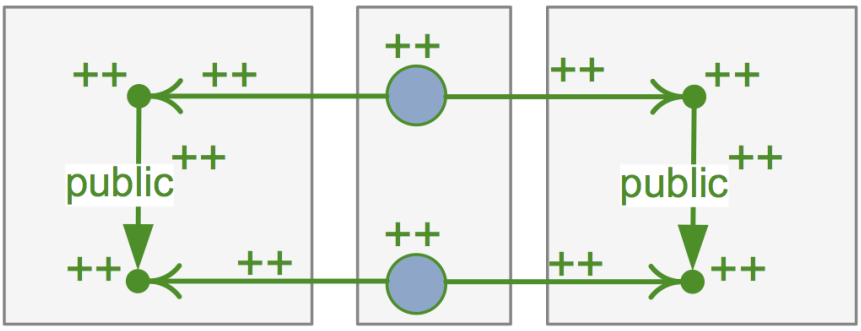


Figure 16: Triple Rule I - public2public

elements with their input, respective output port connected to the gluing elements from the public Reo connector in the IT model. Meanwhile in the business model a filter ABT element is created, corresponding to the two encoding/decoding ABT elements in the IT model. The filter is connected to the input port of the public Reo connector by two other Reo connector. Technically, this means that the messages arriving at the input port of the public Reo connector are delivered to the filter which decides to forward the message to the public Reo connector or to surpress it.

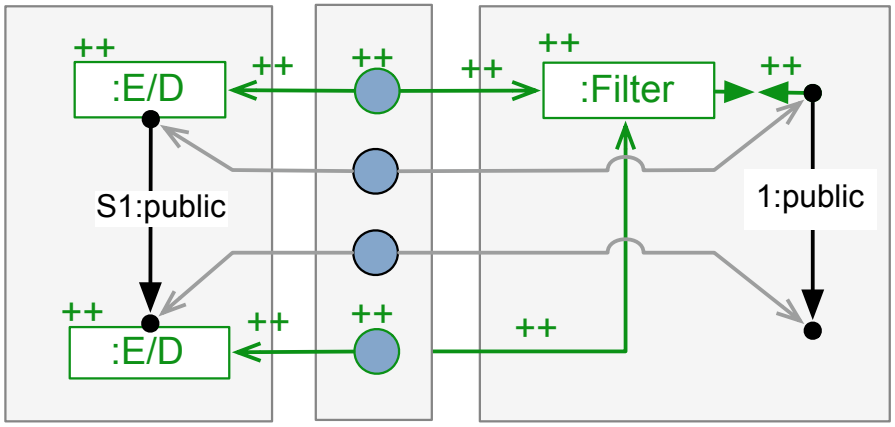


Figure 17: Triple Rule II - E/D2Filter

The triple rule depicted in Figure 18 generates a private Reo connector in between a department and a filter, that is in front of a public Reo connector, if on the corresponding IT model there is a private Reo connector between a LAN element and an encoding element, that is also in front of a public connector.

In Figure 19 the rule is shown, that attaches a public Reo connector with a gluing node to a department ABT element. Note that the department corresponds to the LAN element on the IT domain, that is the destination of the communication line. The notation for the gluing in the concrete syntax with a dashed line and a node labelled “attach” is just used for this rule, as the gluing of ports has no representation in this compact notation. If you

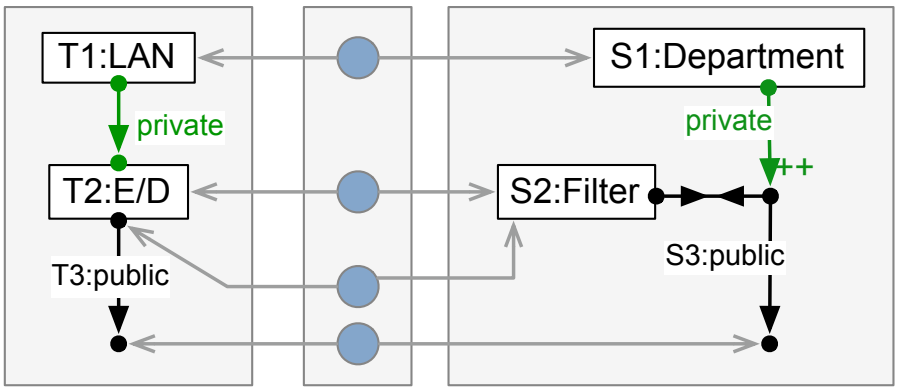


Figure 18: Triple Rule III - PrivateIn2PrivateIn

take a look at the type graph $TG_{ABT-Reo}$ in Figure 11 you can see, that it represents the adding of a edge of the type "glue" between the external input port of the department ABT element and the node with the type " point"

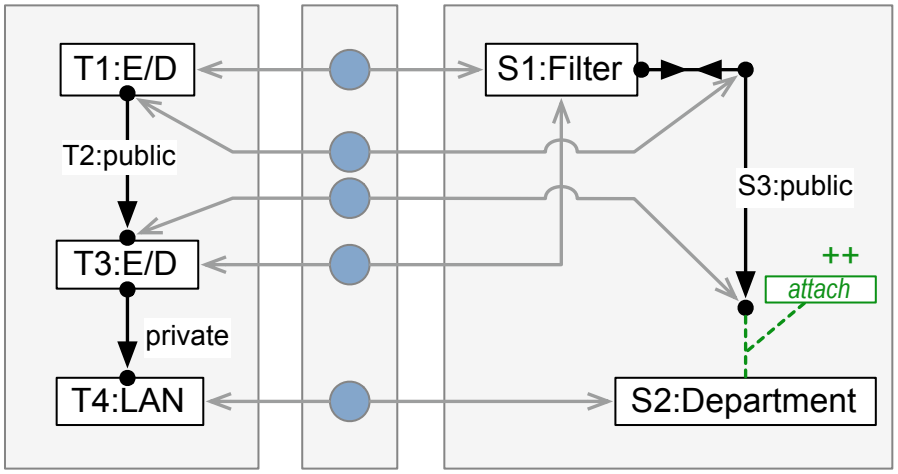


Figure 19: Triple Rule IV - PrivateOut2FilteredOut

5.2 Model Transformation of Models in ABT-Reo

In this section we will transform a simplified model from the one shown in Figure 13 where the communication only works in one direction (see Figure 20) in order to explain to the reader a standard model transformation in ABT-Reo.

The rules in the above section are all triple rules, i.e. they are rules to construct both models simultaneously. As this work is dealing with the subject of model transformation, we need rules to create a model in one domain from an existing model in the other domain.

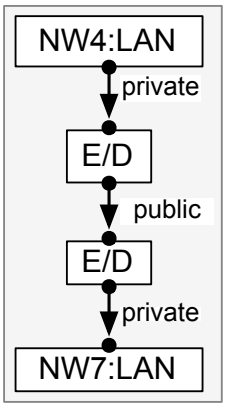


Figure 20: The model before the transformation

Therefore, we need to create forward rules, i.e. rules transforming from the IT to the business domain, derived from the given triple rules. For this purpose we assume, that all the parts that are created by the source rule are already existing in the forward rule in one model, here in the IT model. Figure 21 shows the forward rule created from "triple rule I" in Figure 16. The black colored public Reo connector and gluing points are not created by this rule, but they need to exist in the model to make this rule applicable. In the same way the "triple rule II" in Figure 17 can be transformed into a forward rule, bringing the encoding/decoding ABT elements into the LHS, i.e. coloring them black and removing the double plus signs in the compact notation.

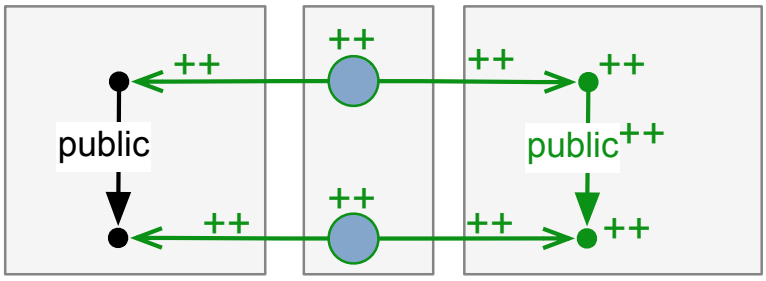


Figure 21: Forward Rule I - public2public

Instead of the compact notation, the forward rule extracted from the "triple rule I" can be represented as shown in figure 22, with the IT model on the left side. We can see the left-hand-side and the right-hand-side of the rule in this notation, which will be more convenient for the application of the transformation step.

The model transformation starts with the model as the source graph of a triple graph with empty context and target graphs. These graphs will be build up by the transformation rules, while the source model will remain unchanged. In Figure 23 the application of the forward rule *public2public* is shown. The rule is on top and the start graph in the bottom

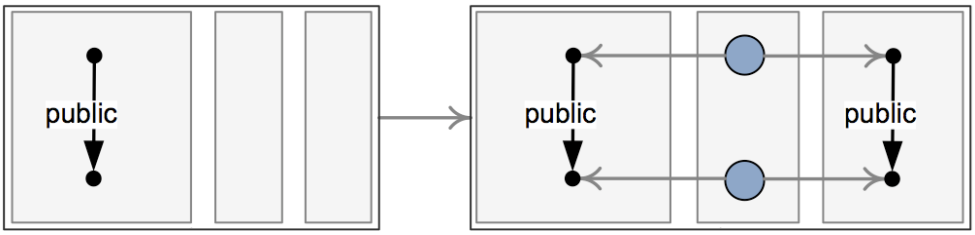


Figure 22: Complete notation of "Forward Rule I"

on the left. The transformation step is a gluing of the start graph and the right-hand-side of the rule over the left-hand-side. Note that the elements, that are identified by the matches are labeled in the inscriptions of the nodes and edges, e.g., the label *S1* on the public Reo connector of the left-hand-side and the same label on the public Reo connector in startgraph means, that the morphism matches these two elements. The result of the transformation step is the graph in the bottom on the right, with the startmodel in the source component and a public Reo connector with two gluing points in the target component with two nodes in the correspondence component and morphisms depicted as grey arrows.

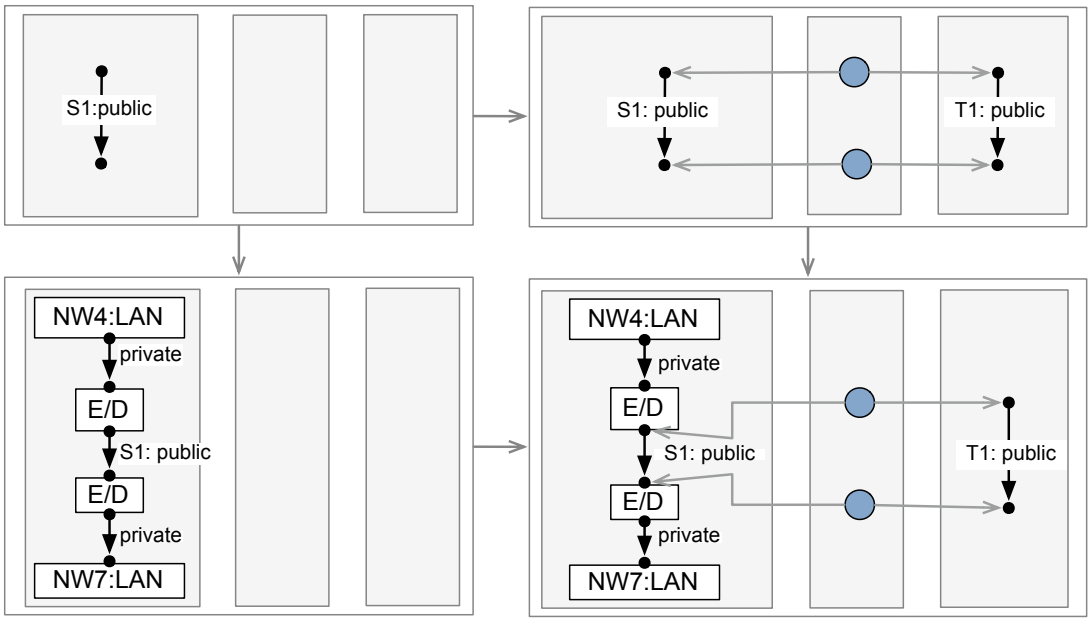


Figure 23: Applying the forward rule public2public

The next transformation step creates a filter in the business domain that corresponds to the encoding/decoding ABT element in the IT domain. Note that we now have the triple graph we just received by the previous transformation step on the left, while above is the rule *E/D2Filter* that is derived from the triple rule in Figure 17.

In Figure 25 we transform the LAN elements with the forward rule *LAN2Department*,

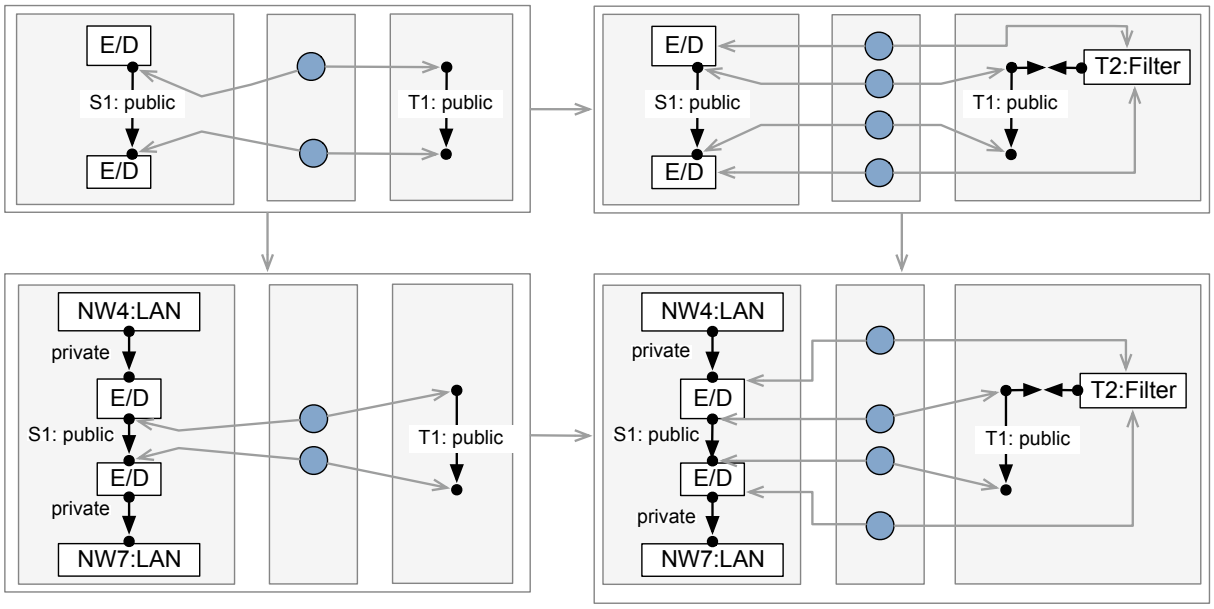


Figure 24: Applying the forward rule E/D2Filter

which is derived from Figure 15. Note that this rule could be at any step before, since it is sequential independent with the rules used in the previous steps (see Definition 13), but it has to be performed in this step at the latest, because the parts, that are generated by this rule in the target component are needed for the following rules to find a valid match.

This rule has to be applied a second time with another match to generate a department node that corresponds to the ABT element *NW7:LAN*. This step is not shown, but works similar to the transformation shown in Figure 25 and the result of this transformation can be seen in Figure 26 as the left graph in the bottom.

In the next transformation step a Reo connector is added, that represents the private communication line outgoing from the department to the public communication line with the filter attached to it. The rule is the forward rule from Figure 18 and the transformation step can be seen in Figure 26.

The last step which completes the transformation of our IT model is shown in Figure 27. It connects the point which is glued to the external output port of the public Reo connector to the external input port of the department element in the bottom of the business model. The attachment, which is shown in the original triple rule in Figure 19, is depicted in this complete notation as the moving of the point to the ABT element, which means the adding of the *glue* edge in the abstract syntax.

In Figure 28 the final result of this transformation is shown. It is an integrated model, which means that both models are connected through the correspondence component in the middle, with the IT model on the left and the business model on the right. To complete the transformation and regain the transformed model, the integrated model can be restricted to the target model. However, leaving it as the integrated model shows the connection

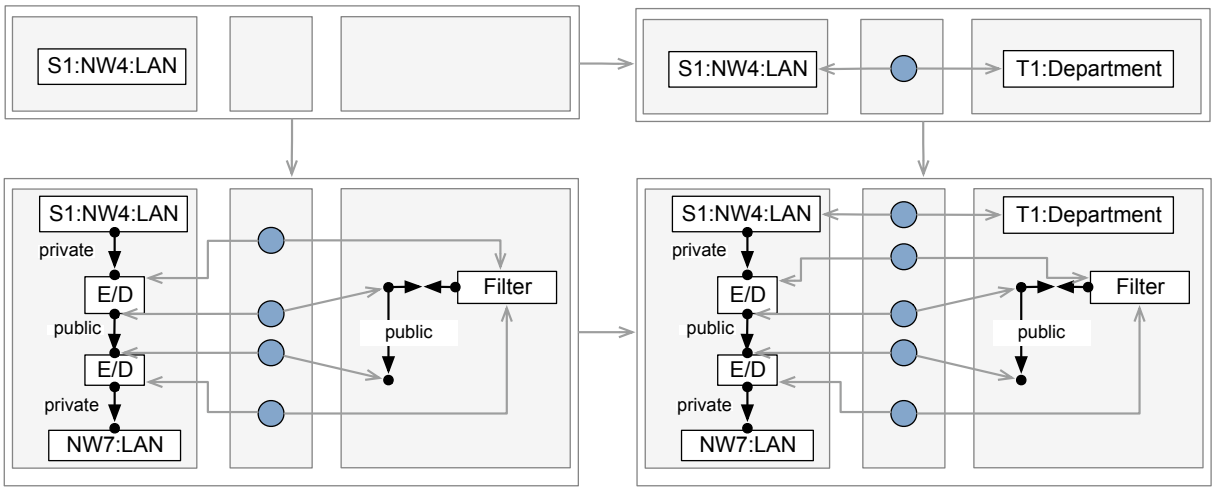


Figure 25: Applying the forward rule LAN2Department

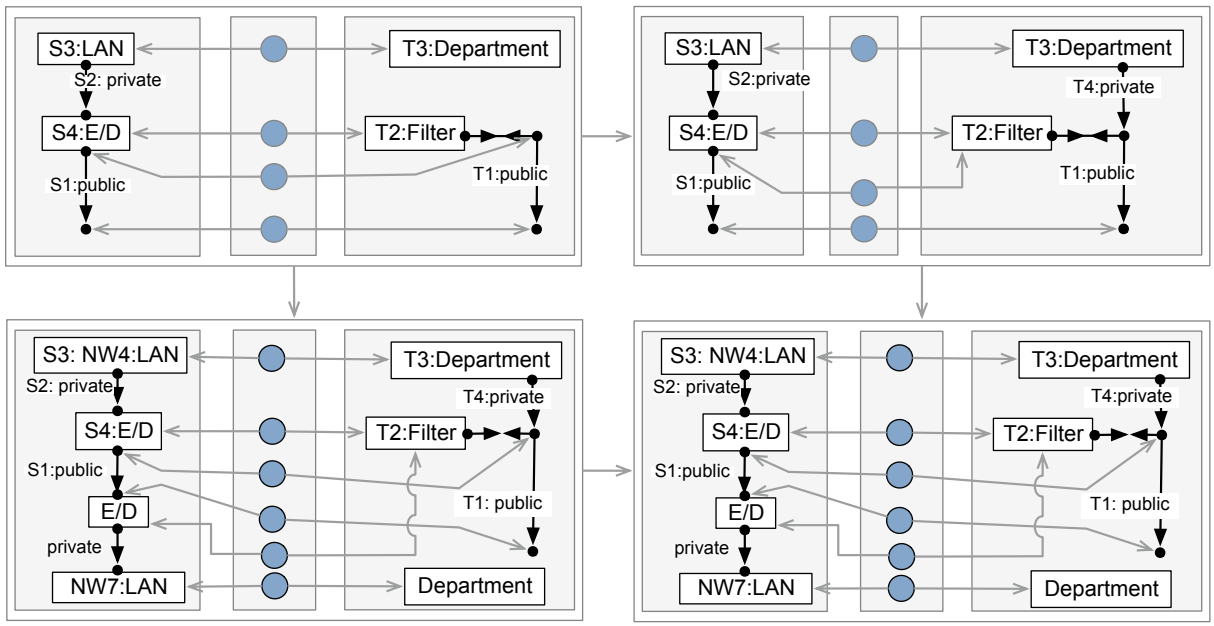


Figure 26: Applying the forward rule PrivateIn2PrivateIn

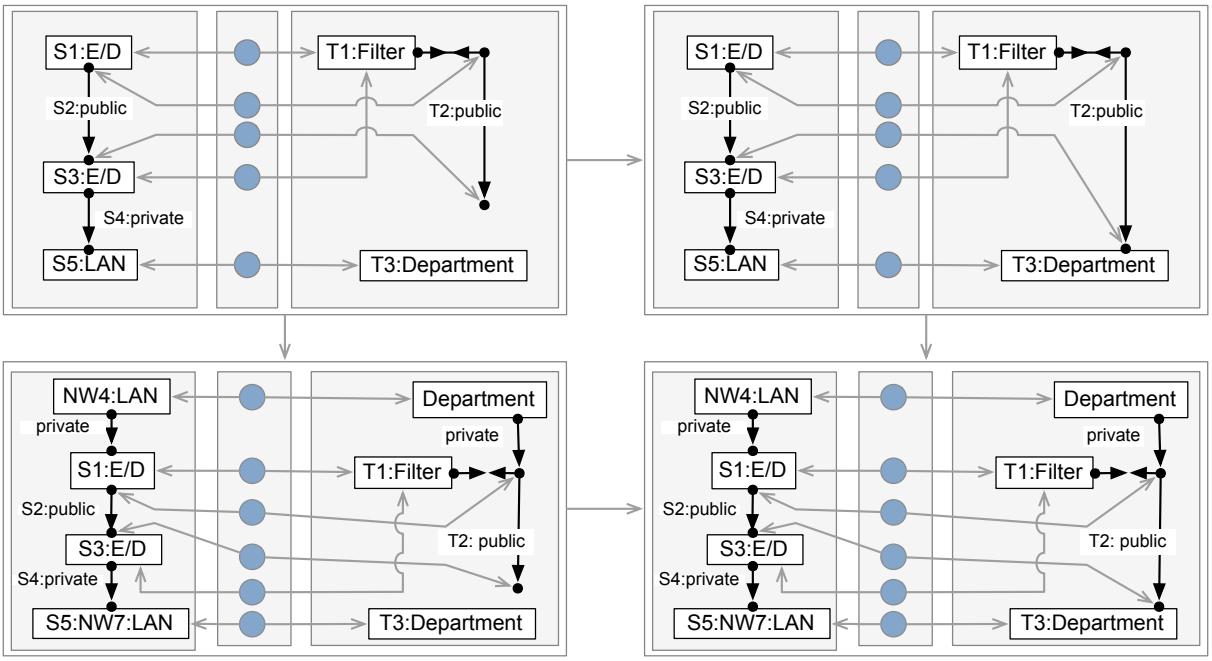


Figure 27: Applying the forward rule PrivateOut2FilteredOut

between the models and moreover, it will be useful in the following section.

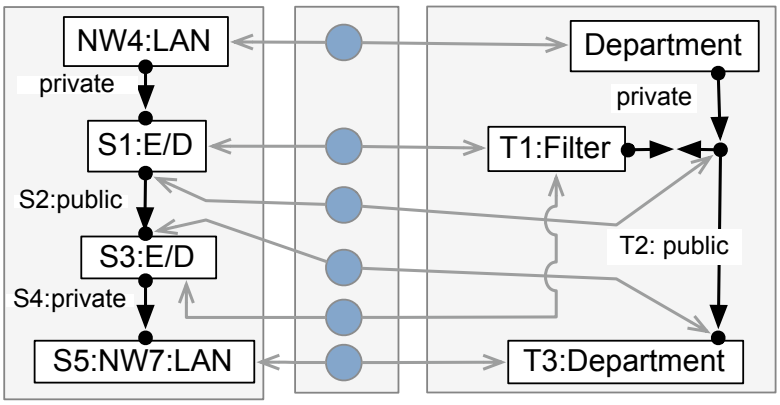


Figure 28: The integrated model

5.3 Model Transformation of Constraints in ABT-Reo

The model fragment we want to transform is a constraint for an IT model. It is shown in Figure 29 in concrete syntax. An IT model satisfies this constraint, iff when it satisfies the premise, it satisfies the conclusion as well. This means if the model is containing a public Reo connector, the ports of this public Reo connector have to be glued to the ports of encoding/decoding ABT elements. This is a constraint for an IT model, our source domain. The premise P is the source component P^S and the conclusion C is C^S in our context. We want to derive a forward model transformation sequence.

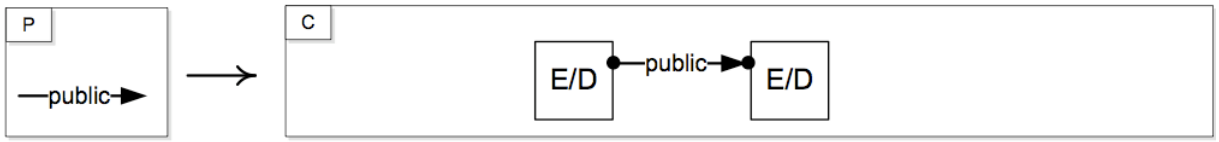


Figure 29: A graph constraint in the IT domain in abstract and concrete syntax

The source graph in the LHS of the "Forward Rule I" in Figure 21 is larger than the premise graph of the constraint, so the borrowed context approach has to be applied, in order to transform this model fragment sucessfully. As shown in figure 30, there is a partial match from the LHS - the upper triple graph in the middle - to the premise - the lower triple graph on the left. The element, that is matched by the partial match, is the public Reo connector. Therefore the morphism $D \to P$ in the span $(P \leftarrow D \to L)$ is the identity. Hence the resulting PO over the span P^+ is the identity of the LHS L and the result of the transformation is the identity of the RHS. Since all parts of the model P have been transformed, the transformation is finished after applying this rule.

Now, the conclusion of the constraint has to be transformed using the same sequence of transformation rules, i.e. using the rule *public2public*, and some matches compatible with the morphism $a : P \to C$. In this case, there is a total match leading to the transformation

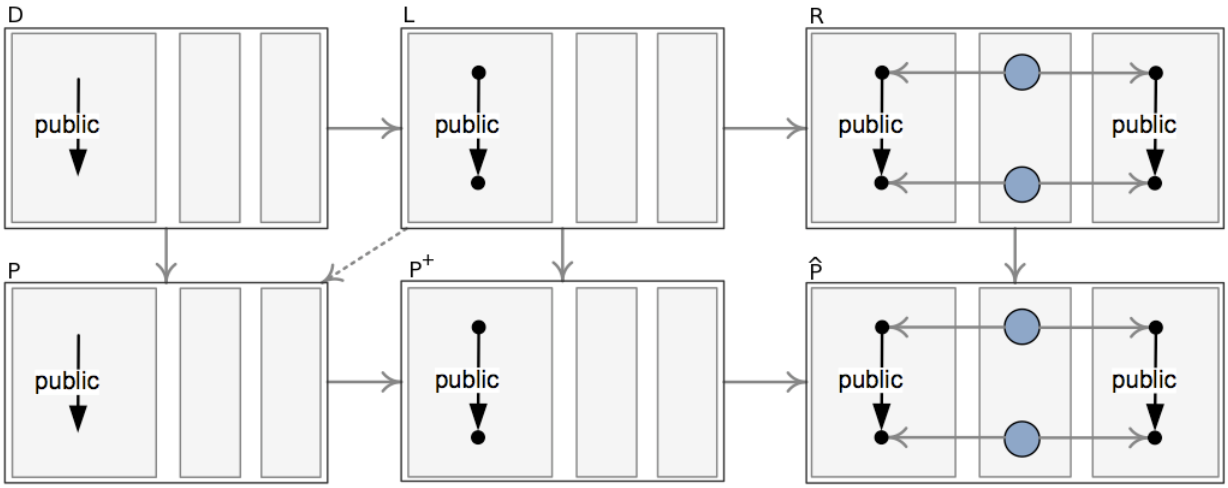


Figure 30: Transformation of the premise of the constraint

step, as pictured in Figure 31.

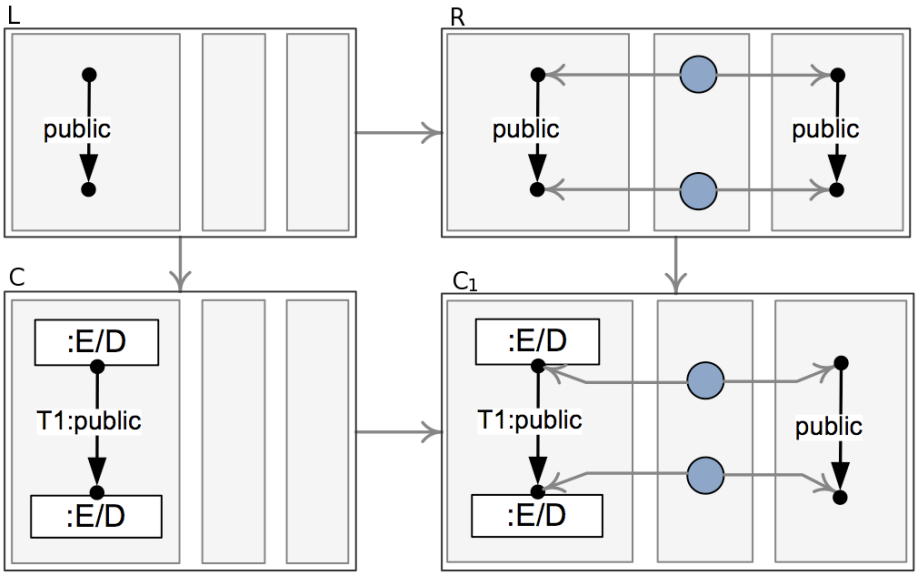


Figure 31: First transformation step of the conclusion of the constraint

In the first transformation step we have to match the public Reo connector from the source component in the LHS to the Reo connector in C^S, since this match is given by the graph constraint.

Furthermore, from Definition 32 we know, that the transformation sequence for C is an extension of the transformation of P shown in Figure 30. The span D' hence is the identity of L and therefore the result is the transformation step without borrowed context shown in Figure 31.

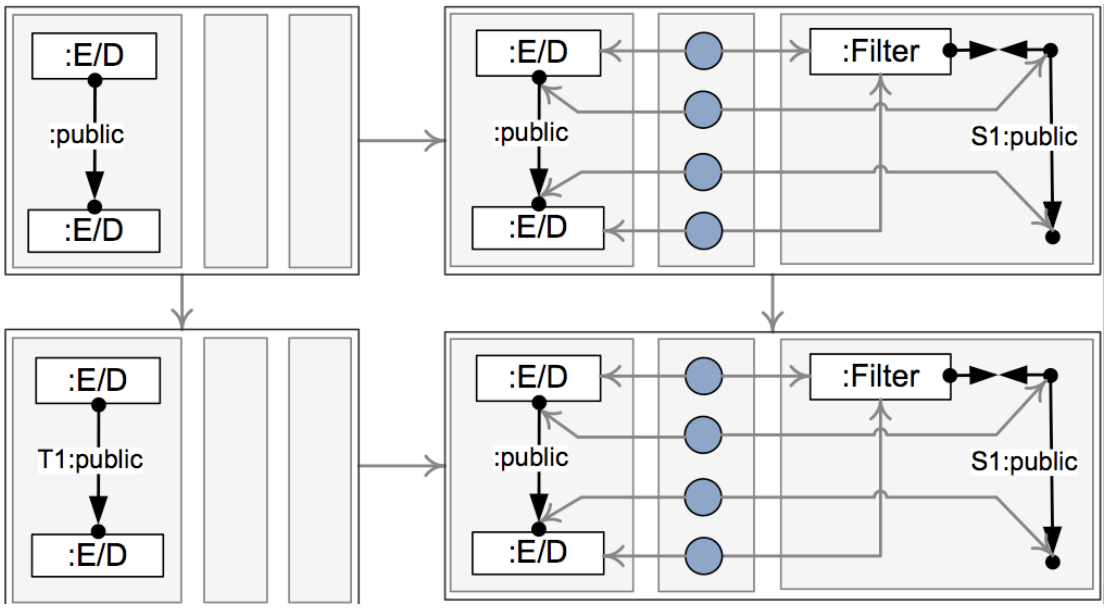


Figure 32: Second transformation step of the conclusion of the constraint

For the next step we use the "Forward Rule II" corresponding to "Triple Rule II" E/D2Filter in Figure 17, which was not used for the premise. So the first possible match is the identity match on the source component. The other possible pastial matches would again lead to inconsistent ABT-Reo models and are neglected.

The resulting constraint in the integrated model IMT is picturized in figure 33. The premise has been extended on the source component from P^S to $\widehat{P^S}$, now also including the gluing points and not only the public Reo connector, while the graph C^S remained unchanged. As explained above, there is only one resulting model for the premise and the conclusion, which is due to the syntax of the ABT-Reo diagrams.

The transformed contraint now states in the business domain, that when there exists a public Reo connector with gluing points to its ports, i.e. there is a connection from the ports, then there has to be a filter before the input port of it. This is from the semantic point of view the representation of the constraint on the IT domain. It does not allow for public Reo connectors to transmit certain messages. There is either no transmission at all or only encrypted communication.

Since we were able to transform the graph constraint now it is interesting if we can apply Theorem 10 which states that if a source model satisfies the propagated source constraint, than the integrated model satisfies the integrated constraint and the target model weakly satisfies the target constraint. The precondition, that in the triple graph grammar ABT-Reo has no critical pairs will be shown later.

We can now use the model from Section 5.2 and show, that according to Theorem 10 that the integrated model satisfies the integrated constraint. Therefore, the source constraint, shown in Figure 20, has to satisfy the source constraint. As shown in Figure

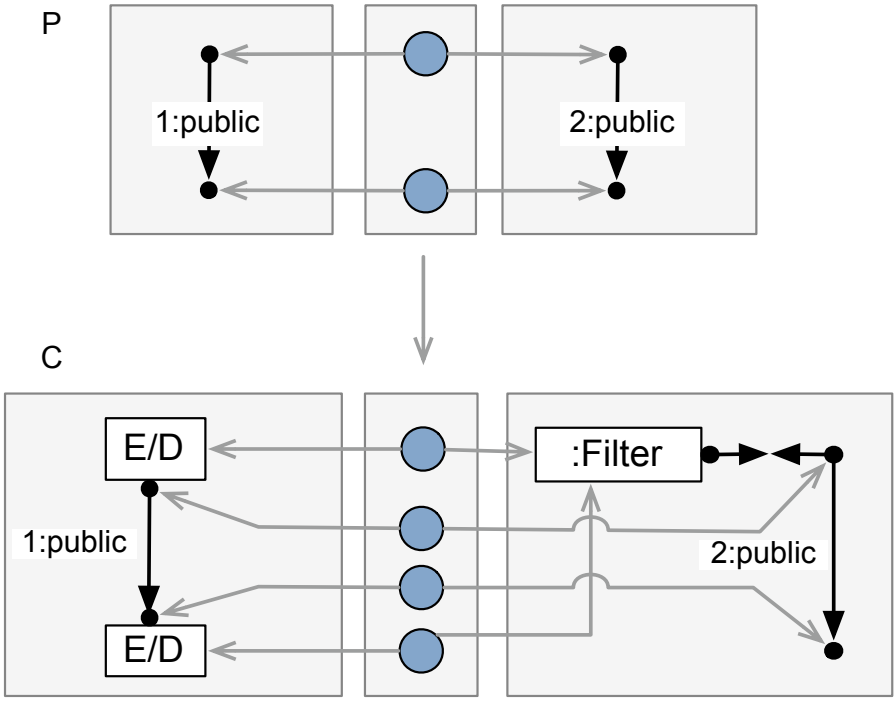


Figure 33: The constraint in the integrated model

34, we can find matches $p : P \to G^S$ and $q : C \to G^S$ such that die diagram commutes, i.e., $p = q \circ a$. In Figure 35 is shown, that the integrated model satisfies the integrated constraint, with the triple morphisms $\widehat{p} : \widehat{P} \to G$ and $\widehat{q} : \widehat{C} \to G$ and the diagram commuting, i.e., $\widehat{p} = \widehat{q} \circ a$. Moreover, the target model weakly satisfies the target constraint, shown in Figure 36. Weakly satisfaction $G^T \models PC(\widehat{a}^T)$ means, that for all injective $p^T : \widehat{P^T} \to G^T$ which can be extended to injective integrated morphism $p : \widehat{P} \to G$ exists injective $q^T : \widehat{C}^T \to G^T$ with $q^T \circ \widehat{a}^T = p^T$. There exists only the injective p^T shown in Figure, which can be extended to the $\widehat{p}$ shown in Figure 35. Now, there is a q^T, which is the target component of $\widehat{q}$, with $q^T \circ \widehat{a}^T = p^T$.

According to the remark after Theorem 10 the assumptions of Theorem 10 are satisfied, if there are requires no critical pairs. By means of the analysing technique in [HEOG10] we can use the tool AGG for the analysis of strong functional behaviour and show, that there are no critical pairs in the derived Forward Translation Rules. In the paper [HEOG10] was shown that in this case for the system with traditional Forward Rules strong functional behaviour is ensured. From this is follows that there is no need for backtracking and hence, every partial source consistent forward sequence has the option to be completed.

In order to analyse the triple graph grammar we reduced the ABT-Reo type graph to the ABT and Reo elements and their connection, the gluing points, ignoring the input and output ports and adding the Strings as attributes instead of additional nodes (see Figure 37). This is a valid simplification, where we left out the ports, because we only use external

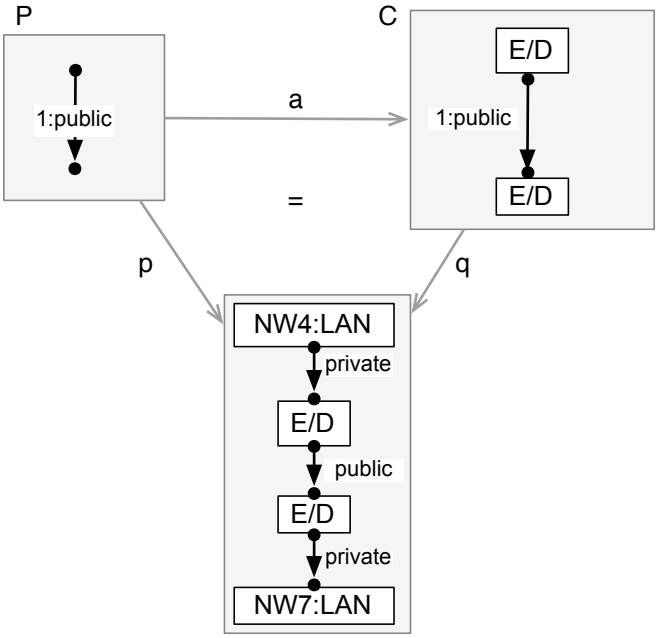


Figure 34: The source model satisfies the source of the integrated constraint

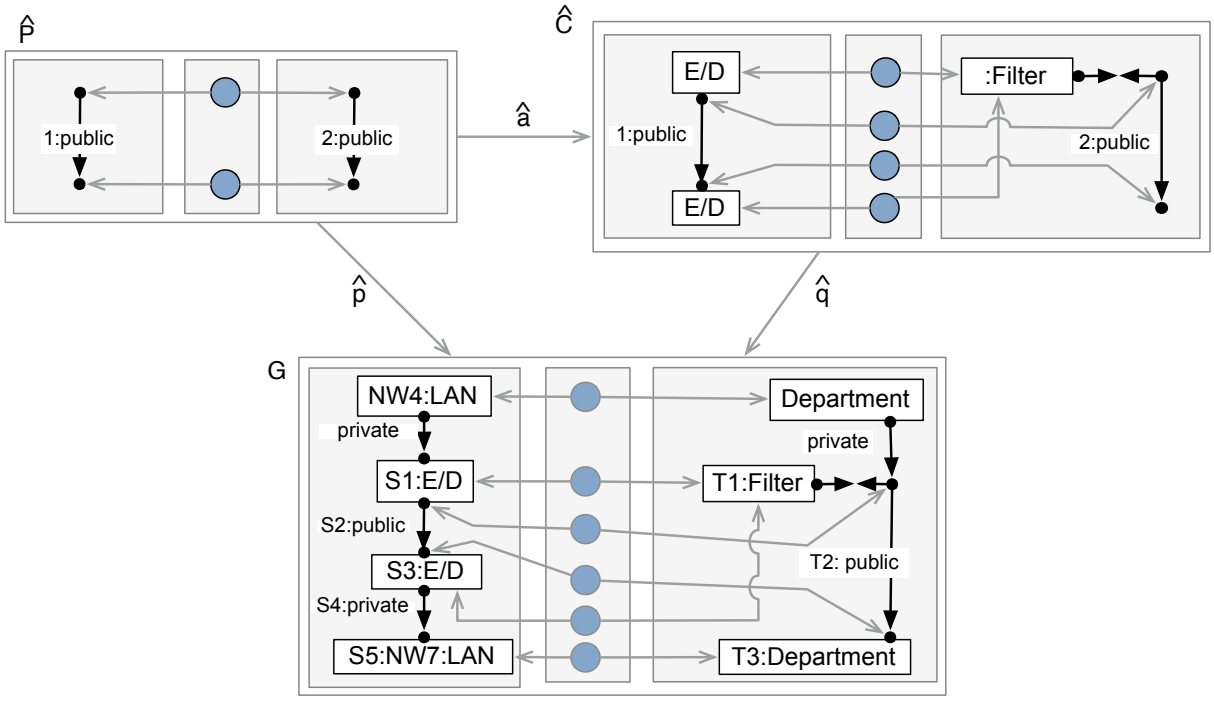


Figure 35: The integrated model satisfies the integrated constraint

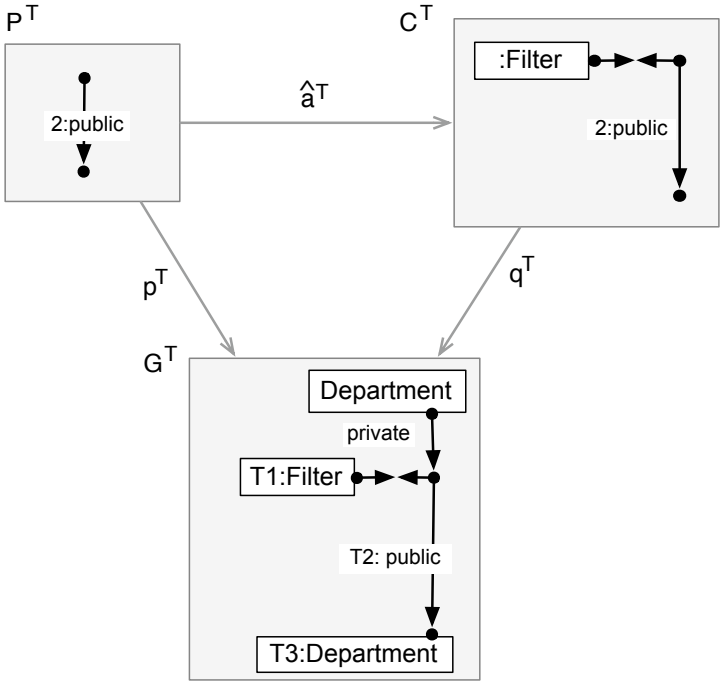


Figure 36: The target model weakly satisfies the target constraint

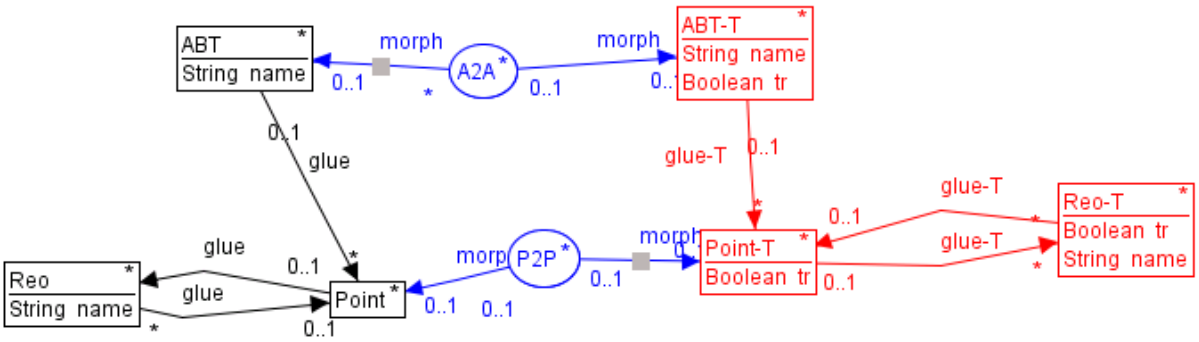


Figure 37: The simplyfied type graph of ABT-Reo in AGG

input and outputports, which are represented by the direction of the edges between the Reo connectors and the points. If a point is the source of a glue edge to a Reo connector, this means it is glued to its input port and, if a ABT element is connected to this point as well, it is connected to its output port and vice versa. AGG supports only plain graphs, but the analysis results are also valid with the notation used in the pictures below. On the left side there is the black colored business model and on the right side is the red colored IT model. In between is the blue colored correspondence graph with blue colored arrows representing the graph morphisms.

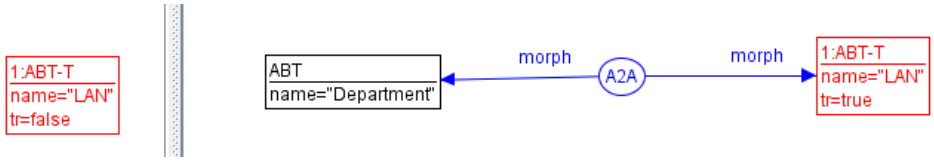


Figure 38: The Rule *LAN2Department_BW*

This reduction was done, because AGG generates many overlaying graphs when analysing critical pairs with the complete model. Analysing the simplified graphs for critical pairs generates significant less overlaying graphs, but we will see later, that AGG will have problems at some point in the analysis though.

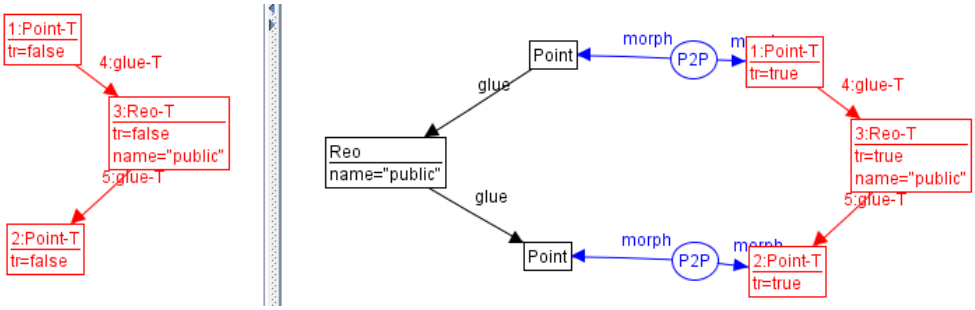


Figure 39: The Rule *public2public_BW*

In Figure 39 the "Forward Rule I - *public2publice*" from Figure21 is depicted, which transformes a public Reo element with gluing points in the IT model into a corresponding public Reo element with gluing points in the business model. Because in AGG ABT-Reo was implemented with the business model in the source and the IT model in the target component, it is a backward rule, which makes in our case no difference for the analysis.

The rules picturized in the Figures 38 and 41 to 42 are representing rules from the ABT-Reo triple graph grammar given by the triple rules in Figure 15 and 17 to 19.

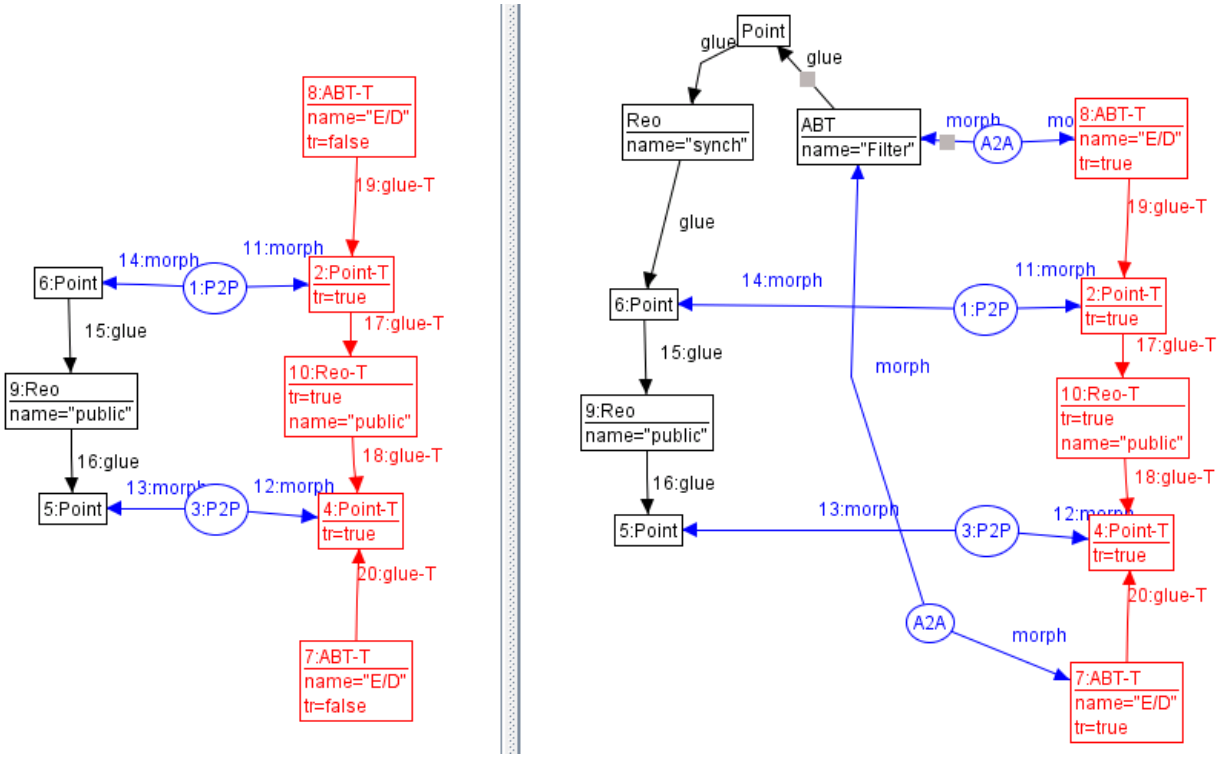


Figure 40: The Rule *Filter2ED_BW*

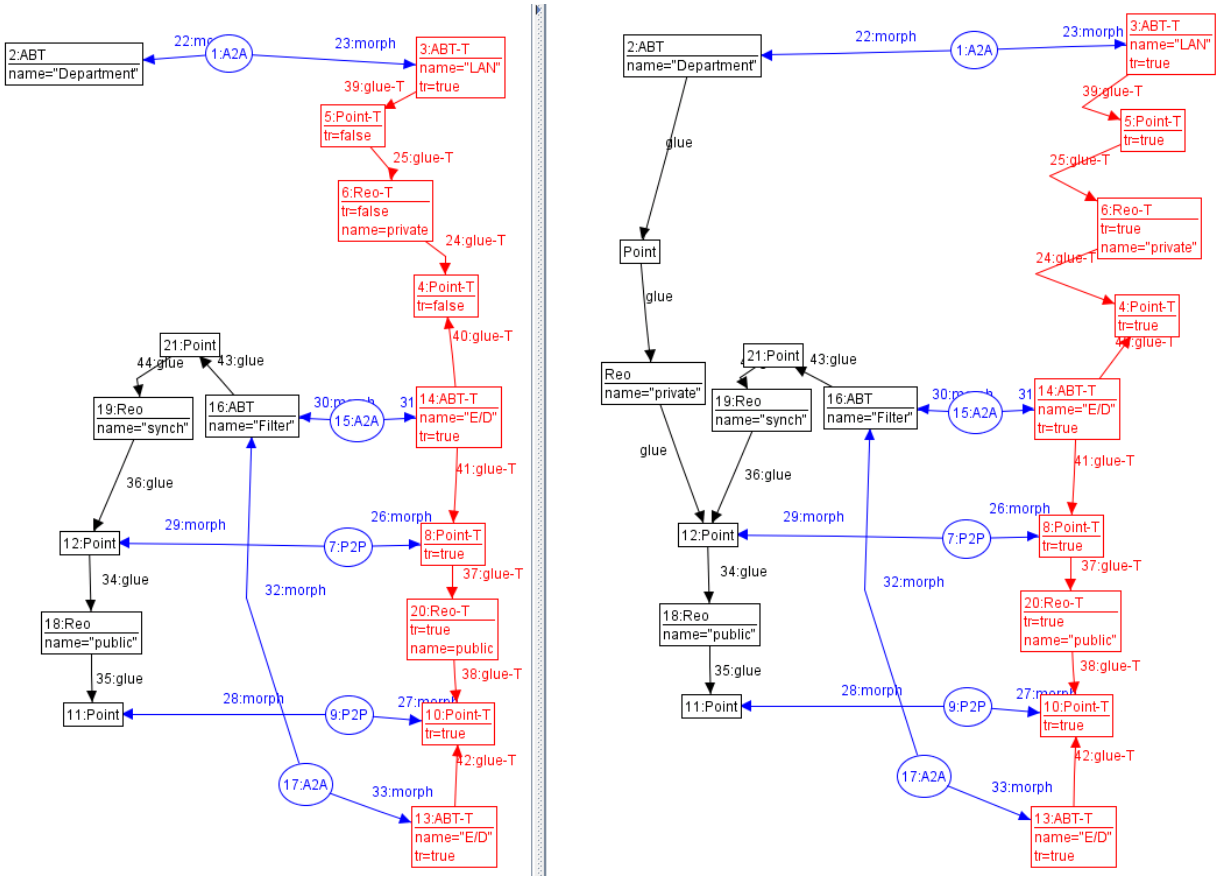


Figure 41: The Rule *PrivateIn2PrivateIn_BW*

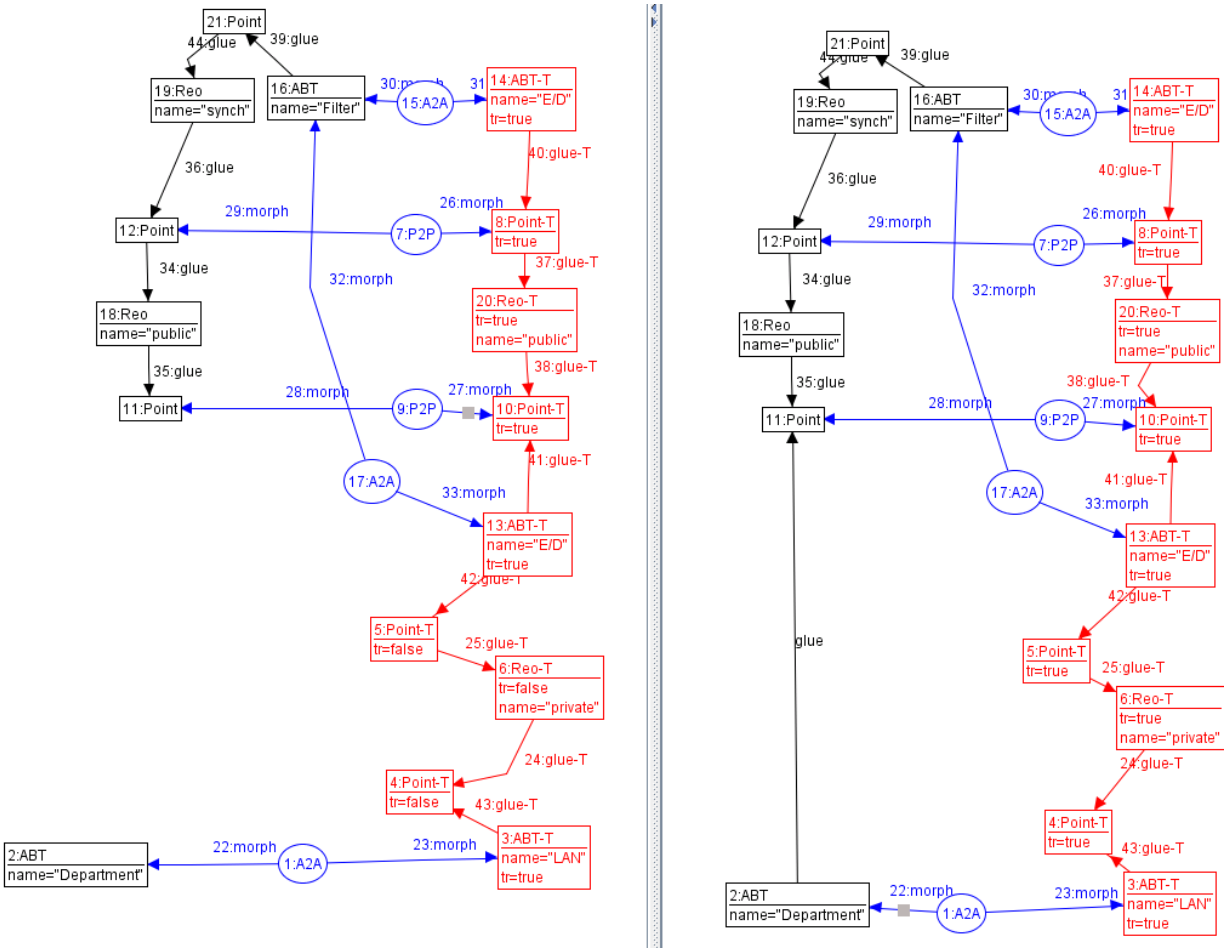


Figure 42: The Rule *FilteredOut2PrivateOut_BW*

In order to exclude graph structures, that cannot be generated by the source rules, but are allowed by the type graph, we added graph constraints to the grammar in AGG. The premise in the screenshots is on the left hand side, while the conclusion is on the right. Moreover, since we want to forbid such structures, the graph constraints are negated, i.e. if there is found a match from the premise into a graph, there must not exist a match from the conclusion into the graph, such that the triangle over the morphism from the premise to the conclusion commutes.

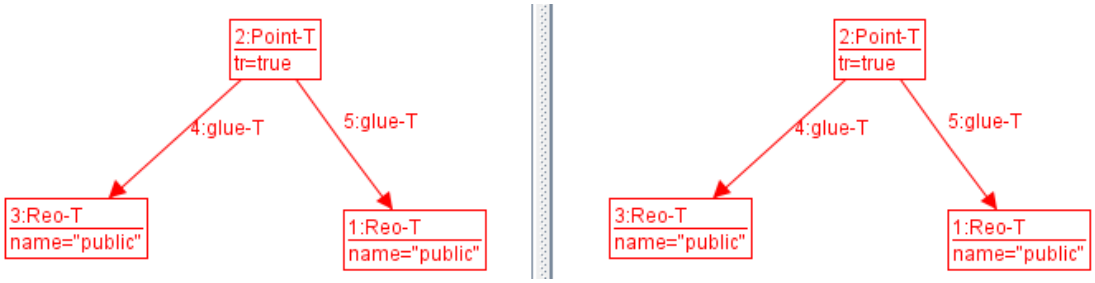


Figure 43: The constraint forbids two glued "public" Reo elements

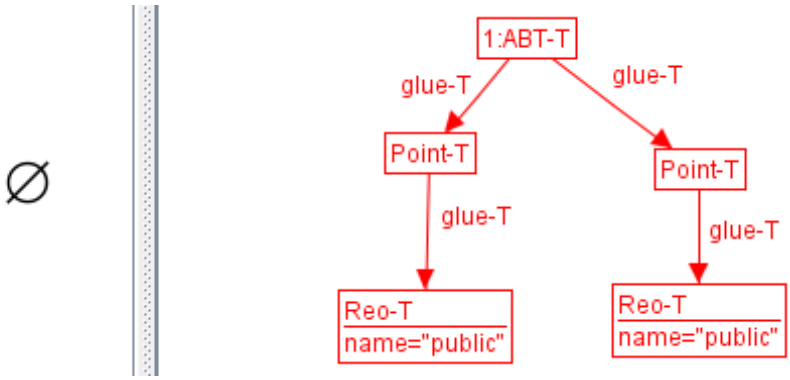


Figure 44: The constraint forbids an ABT element glued to two Reo elements

The constraint in Figure 43 forbids the existence of two public Reo connectors that are glued together. In Figure 44 the premise is empty, which means, that the conclusion must never exist in any model. Here, that there is no more than one public Reo connector glued to an ABT element. The third constraint, shown in Figure 45, forbids any gluing point to glue the input port of a private Reo connector to any input port of another Reo connector. So practically, these constraints ensure the right gluing of Reo connectors.

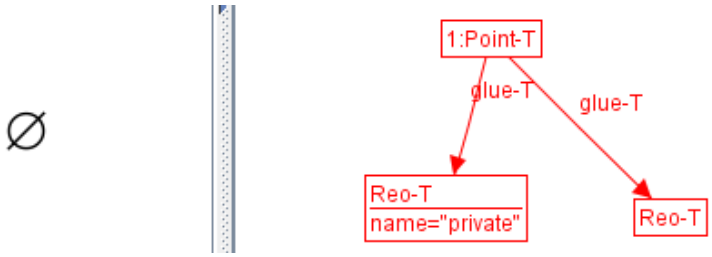


Figure 45: The constraint forbids a "private" Reo element to be glued to another Reo element

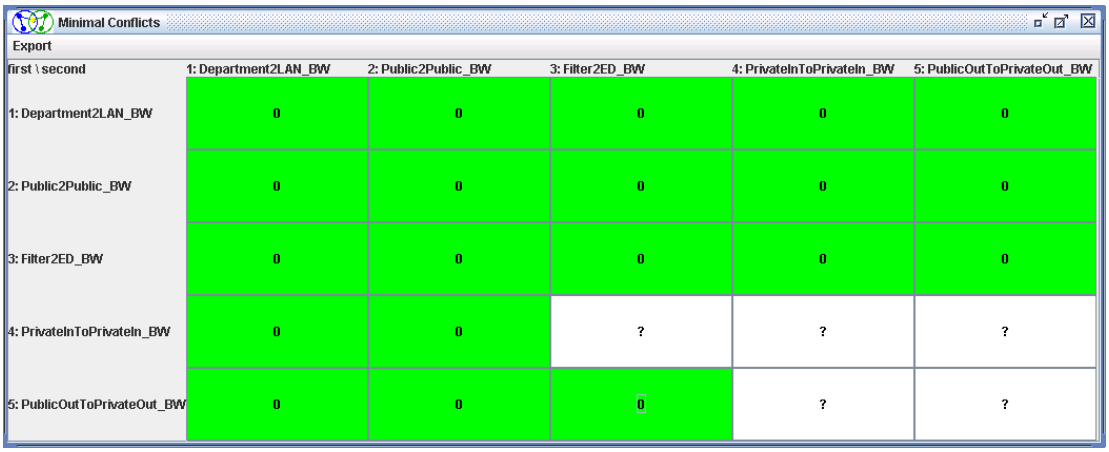


Figure 46: The critical pair analysis in AGG

The result of the critical pair analysis for the given ABT-Reo Forward Rules is shown in Figure 46. The green cells in the table indicate, that there are no critical pairs for the two analysed rules. The analysis works in two dircetions, because AGG computes the delete-use-conflicts between two rules, where the first rule deletes elements, i.e. changes attributes, while the second rule needs those elements. In row four and row five we can see question marks, which are indicating, that AGG could not complete the analyses for these rules. This happens, when AGG runs out of workspace with over 500,000 overlapping graphs and cancels the analysis, which can happen in a delete-use-conflict in one way, but eventually not the other. However, we can argument, why there are no critical pairs for the remaining rules.

First of all, we have a triple graph grammar without NACs (Negative Application Conditions). Therefore, we never encounter the problem, that one rule generates content in the models, which another rule forbids in order to be applicable, i.e. there are no produce-forbid conflicts. So at the most we get so-called delete-use conflicts, i.e. one rule deletes parts of the model, which prevents from finding a valid match for another rule. However, the rules are only deleting on the translation attributes which indicate, if an element of the source model was already translated by changing the attribute from "false" to "true".

Therefore, we will argument that those conflicts with "true" and "false" cannot occur in the triple language ABT-Reo, i.e., one rule does not change elements that are needed for another. The first critical pair analysis that could not be computed was between the two rules *Filter2ED_BW* and *PrivateInToPrivateIn_BW*. The first rule changes the attribute of two E/D nodes from "false" to "true", while the second rule changes the attributes of a private Reo connector and its gluing points.

Moreover, the overlappings between the rules *PrivateInToPrivateIn_BW* and *FilteredOut2PrivateOut_BW* are no critical pairs. The first changes the attributes of a private Reo connector and its gluing points, while the second changes the attributes of a public Reo connector respectively. The constraints shown in Figures 43 and 45 forbid the existence of more than one gluing point connected to a Reo connector and vice versa. Therefore the two critical parts cannot overlap.

Furthermore, AGG could not analyse the critical pairs between the rule *PrivateInToPrivateIn_BW* itself and *FilteredOut2PrivateOut_BW* itself. We ignore overlapping matches, that are identical and for every not identical match, there is no overlapping on the elements where the attributes are changed. In the example of the rule *PrivateInToPrivateIn_BW* the constraint in Figure 45 forbids a point to be glued to a private Reo connector and another Reo connector at the same time, so another match for one of the points or the Reo connector with the attribute false cannot be found, other than the identical.

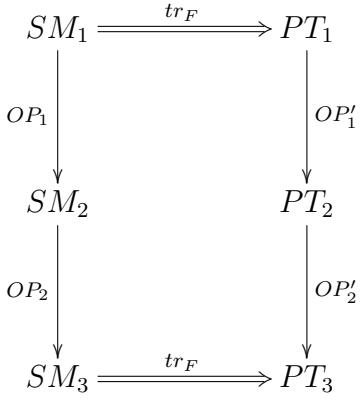
The same holds for *FilteredOut2PrivateOut_BW*, where the attribute of a public Reo connector and its gluing points are set from "false" to "true". Here the constraint from Figure 43 forbids the existence of two public Reo connectors connected to one point. Hence, the identical match would be the only critical pair.

In this case study the use of the borrowed context approach gave us the opportunity to transform model fragments, which we could not before. Additionaly, we were able to get only one single transformation result in this particular case study, which leaves us with an integrated model, where we can use the target graph only as well, i.e. without the source and context graph. Furthermore we can say, that every source model, if under the side condition that the model transformation has strong functional behaviour (see Theorem 10) satisfies the propagated source constraint, then the integrated model will satisfy the integrated model and the target model weakly satisfies the propagated target constraint.

5.4 Model Transformation of Operational Semantics: From State Machines to Petri Nets

Another area of application for the transformation with borrowed context is the transformation of operational semantics. As rules they contain fragments of models which define operations in a model language, e.g., petri nets. For the elementary petri nets there has been defined an operational semantics already [Rei84].

The aim is to achieve a transformation of the operationals semantics such that the diagram below commutes.



In order to define operational semantics for state machines, the abstract syntax for state machines from [EEPT06] has been modified as shown in Figure 47. The graph was reduced to states and transitions linked to a certain statemachine. The transition can trigger an event or can be connected over an action to it. Additionally, a transition can be connected to a state over a condition, which express, that for the transition to be executed, the state has to be active.

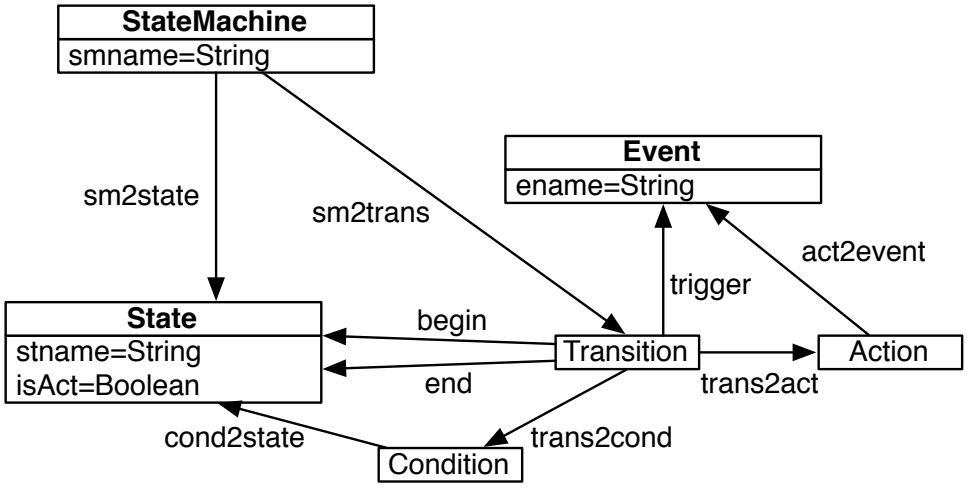


Figure 47: The type graph for state machines

In Figure 48 the type graph for an elementary petri net is shown, with the token as boolean attribute. A transition can be executed, if all places that are connected to it with a prearc must have this attribute as true and all connected to it with a postarc must have this attribute as false. The execution of a transition sets the token attribute of its connected places to false if they are connected by prearcs and to true if they are connected by postarcs.

We will now give the outlook to a case study, that need to be further extended on the transformation of operational semantic. Therefore, we will introduce an example of a

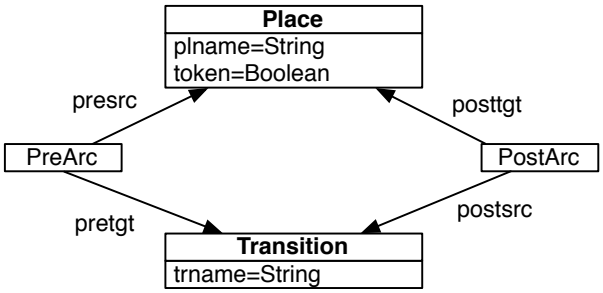


Figure 48: The type graph for petri nets

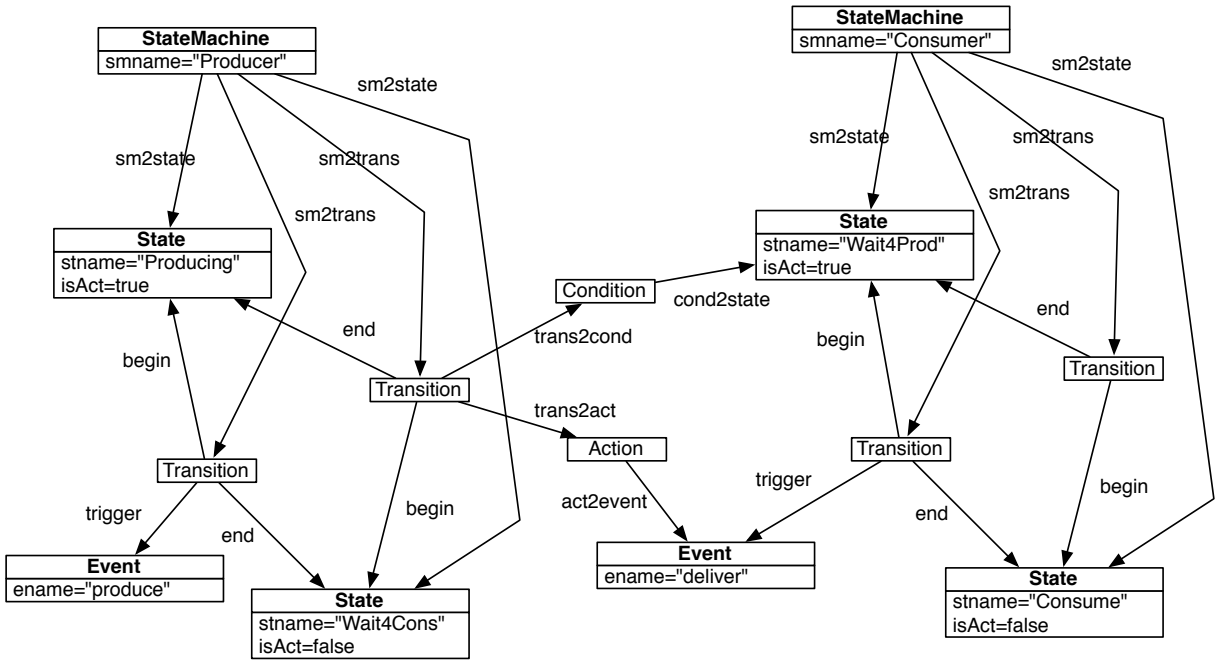


Figure 49: The Producer-Consumer SM in abstract syntax

simple producer consumer state machine in abstract (see Figure 49) and concrete syntax (see Figure 50).

In this state machine we have two machines, a *Producer* and a *Consumer*. The producer starts in the state *Producing*. If he executes the transition going to *Wait4Cons*, the event *produce* is triggered. To execute the next transition, the consumer has to be in the state *Wait4Prod*, and both transitions are executed simulaneously. Then the consumer can consume and so on.

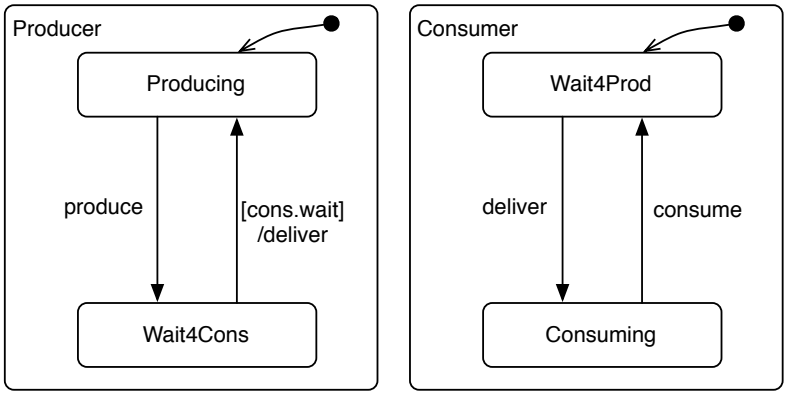


Figure 50: The Producer-Consumer SM in concrete syntax

In order to perform a model transformation from state machines to elemental petri nets the transformation rules developed by [EEPT06] were modified and translated into forward triple graph rules.

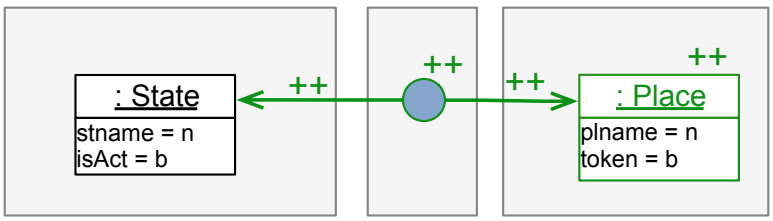


Figure 51: Forward Rule *state2place*

In Figure 51 the rule is shown, which transforms a state from a state machine into a place from a petri net and sets the attributes to the attributes of the respective state. Similiar to that, in Figure 52 an event is translated to a transition. In Figure 53 and 54 the pre and post arcs between a place and a transition is added corresponding to the transition between the state and the event in the state machine. If the transition is connected to an event via an action, the forward rules shown in Figure 55 and 56 are to be used. Due to the fact, that the condition, that the state, that is connected over the condition, has to be activated, we decided to translate it in a manner, that is has to be connected to the

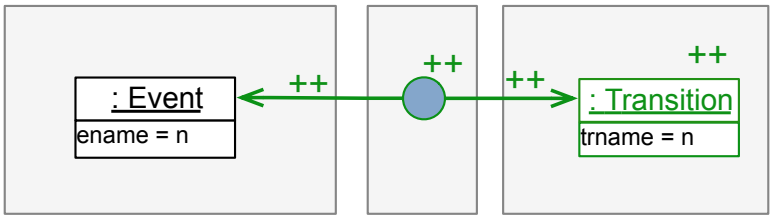


Figure 52: Forward Rule *event2trans*

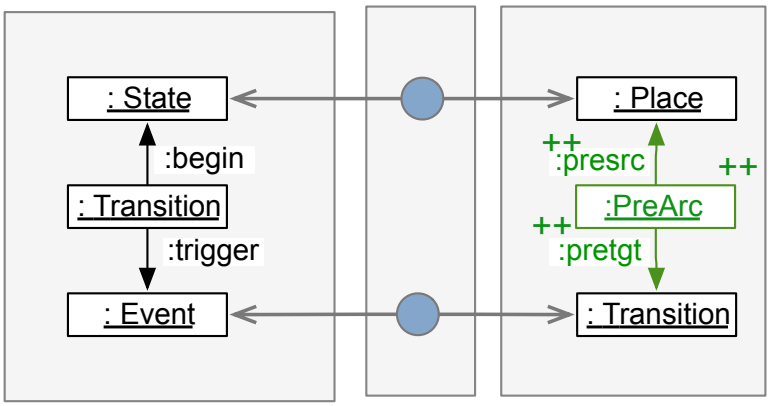


Figure 53: Forward Rule *trans2prearc*

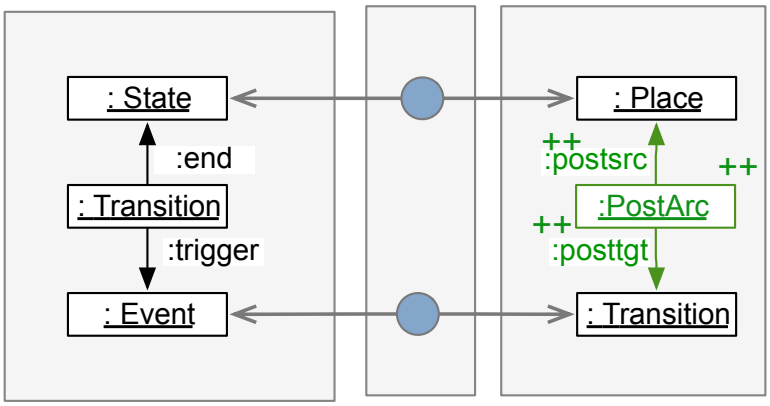


Figure 54: Forward Rule *trans2postarc*

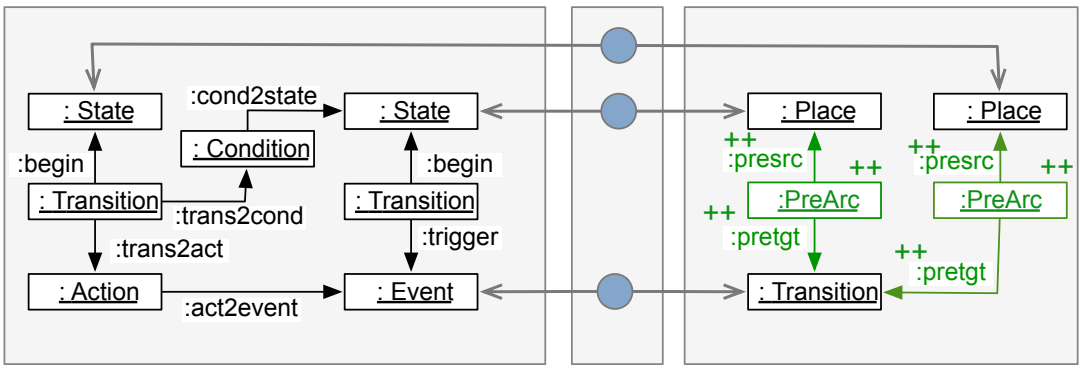


Figure 55: Forward Rule *transAct2prearc*

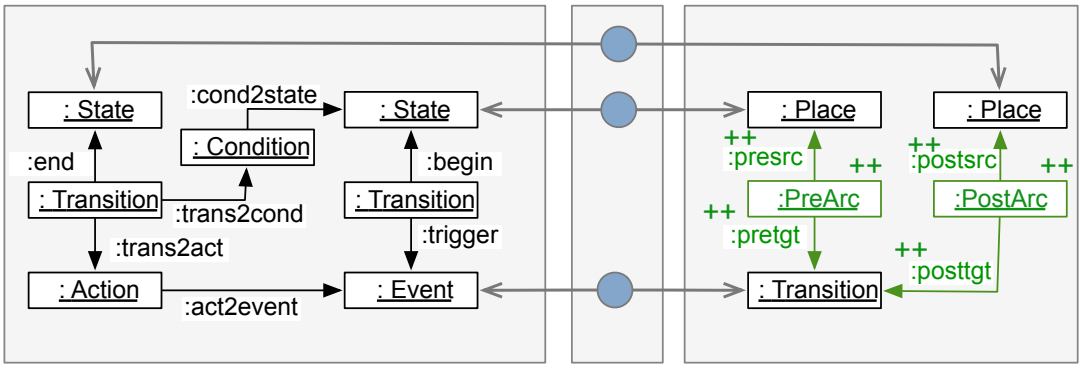


Figure 56: Forward Rule *transAct2postarc*

transition in the petri net domain. Another possibility is to add another transition, that has a prearc and postarc to the place, that must have a token. This was done in [EEPT06], but because of definition of petri nets this transition can never be executed, since the place in the post domain must not have a token. Therefore, we decided for this simple checking method, which possibly can be extended in a more sufficient way.

When we want to define the operational semantics, we encounter the problem, that more than one transition can be executed simultaneously. The most simple case is a transition between two states that triggers an event, without affecting another transition. This simple case would be represented in the rule for operational semantics *directTransition* shown in Figure 57, where the state, where the transition begins, is set from true to false and the state where it ends from false to true.

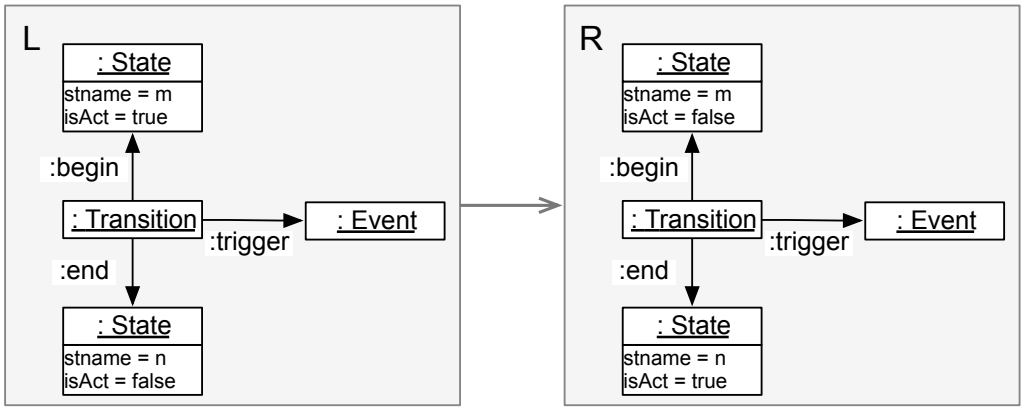


Figure 57: Rule *directTransition* for the Operational Semantics in SMs

This operational semantics rule can be translated with the given rules without problems, i.e., without the use of borrowed context. The result of the transformation is given in Figure 58 and 59, where on the predomain the token is set from true to false and vice versa in the post domain. Note, that this operational semantics rule does not check, neither in source model, not in the target model, whether there are more states, resp. places, are affected by the event, resp. transition.

This can be prevented by priorisation, i.e., that rules with a higher priority are executed until they cannot be applied any furhter. This means, that the rule shown in Figure 60 would be applied, until all transitions connect via actions are executed and after that the rule in Figure 57 would be applied, such that all states affected by the event are changed.

The rule shown in Figure 60 sets the isAct attribute from a state in the beginning of a transition from true to false and in the ending from false to true, if the state in the condition is activated and the transition is connected via an action to an event.

For the transformation of this rule transformation steps with borrowed context would be needed, as shown for the application of the rule *transAct2preArc* in Figure 61.

The transformation of the operational semantics rule *indirectTransition* with the use of borrowed context results in the Figure 62 for the left-hand-side and in the Figure 63 for

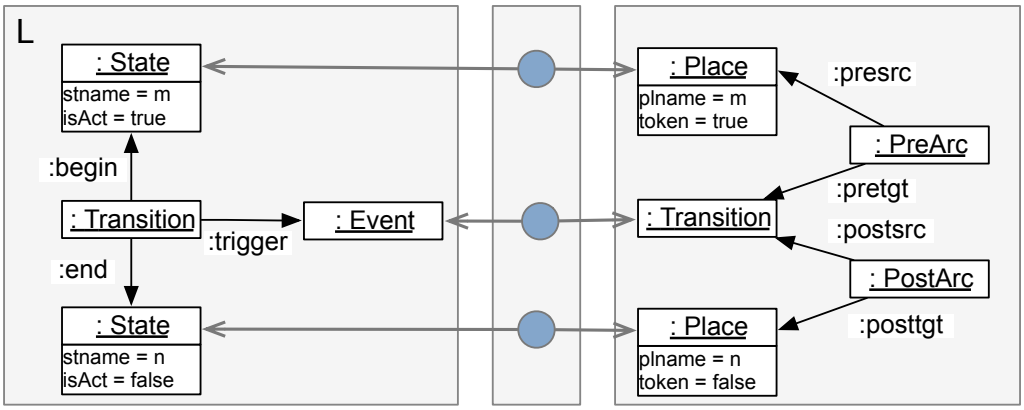


Figure 58: Integrated Model of the LHS from *directTransition*

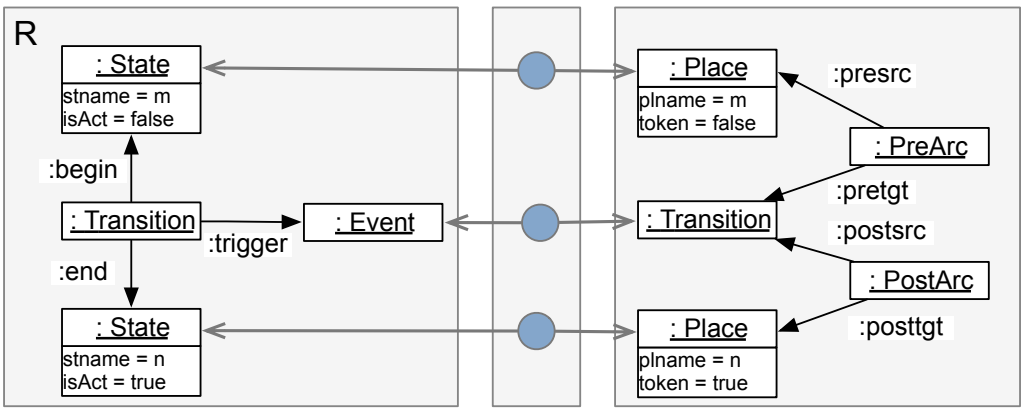


Figure 59: Integrated Model of the RHS from *directTransition*

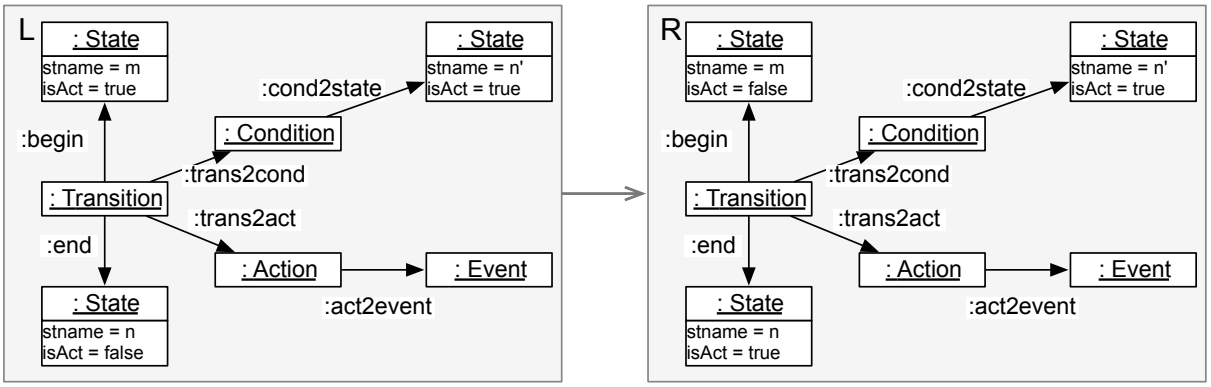


Figure 60: Rule *indirectTransition* for the Operational Semantics in SMs

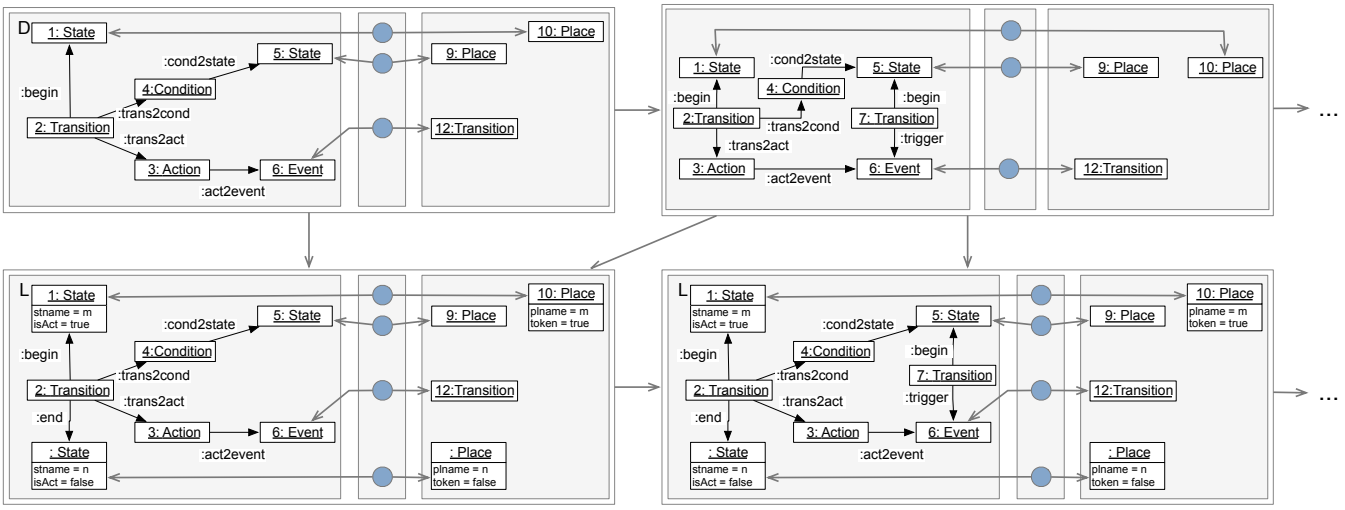


Figure 61: Transformation Step with BC for *indirectTransition*

the right-hand-side.

In order to show that for our state machine seen in Figure 50 we can achieve our goal that we can perform the operational semantic and the transformation independent, we first need to apply the model transformation on our state machine, resulting in the petri net shown in abstract syntax in Figure 64 and in concrete syntax in Figure 65.

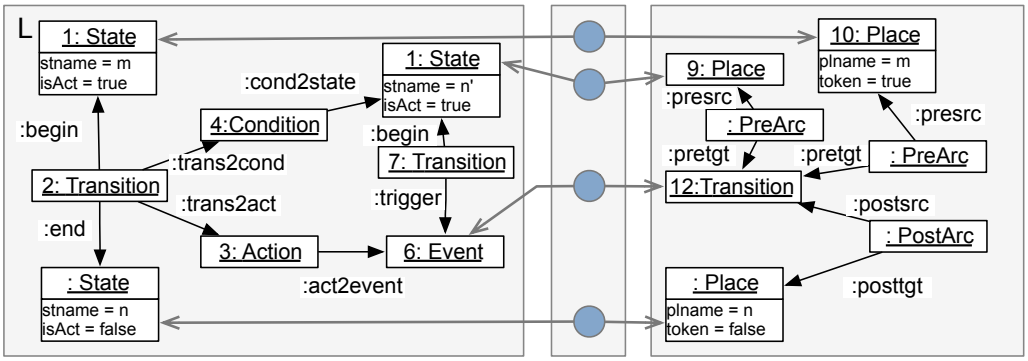


Figure 62: Integrated Model of the LHS from *indirectTransition*

In the following we will give a short overview on the transformation between the models and the execution of the operational semantics. We will use the respective source or target component ot the integrated model of the operational semantic rules and show in this example, that the ways are equivalent, if we first perform it on the state machine and then transform it or, if we first translate the state machine and afterwards perform the operational semantics on the petri net.

1. If we first perform the operational semantic to Figure 50, we have first to apply the

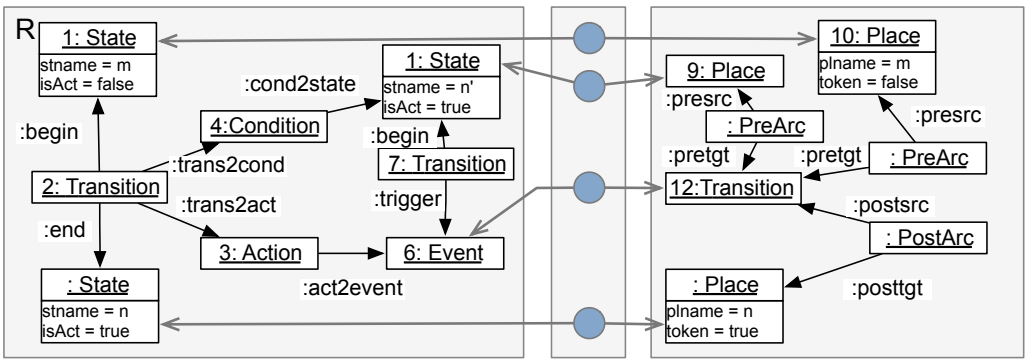


Figure 63: Integrated Model of the RHS from *indirectTransition*

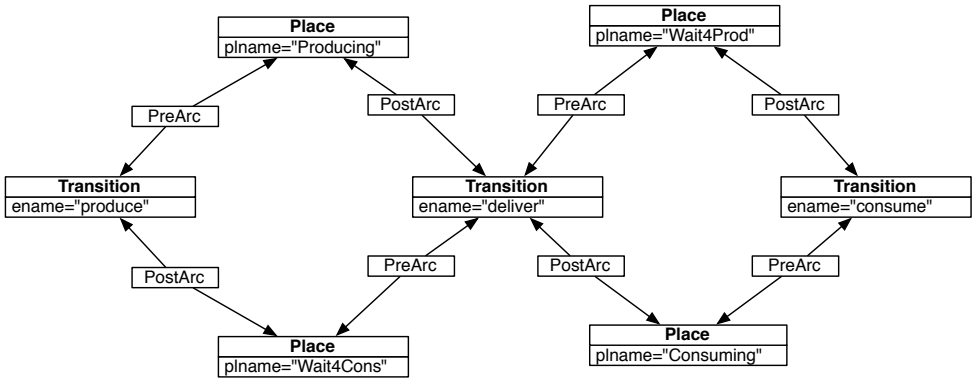


Figure 64: The Producer-Consumer PT net in abstract syntax

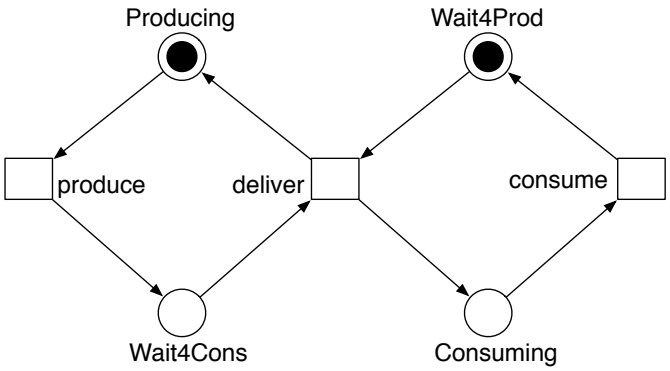


Figure 65: The Producer-Consumer PT net in concrete syntax

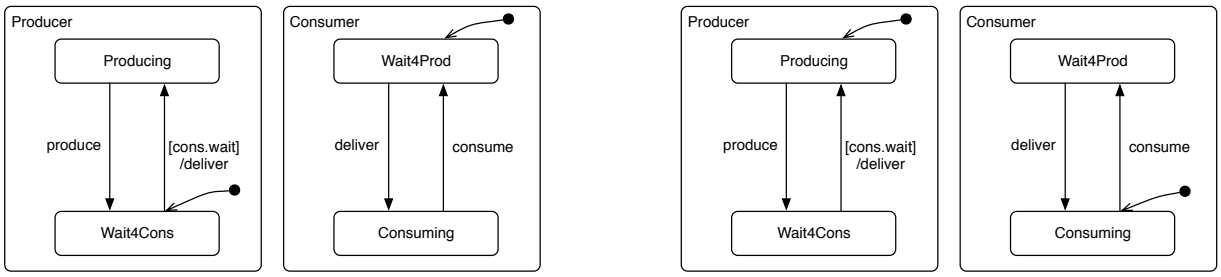


Figure 66: The Producer-Consumer SM after step 1 and step 2 of OP

rule *directTransition*, which is the same in the integrated model, since nothing was borrowed on the source component. This would result in the left model shown in Figure 66, where in the Producer state machine the active state was changed from *Producing* to *Wait4Cons*.

After that we need to first apply the rule *indirectTransition* as long as possible, in this case once. We apply the rule from the source component of the integrated model shown in Figure 62 and 63. This would result in the Producer state machine to switch the active state back to *Producing*. Since this is not yet a valid operation, we still need to perform the rule *directTransition*, which results in the state machine on the right in Figure 66.

Finally, we transform the state machine with the given rules. This results in the elemental petri net shown on the right in Figure 67.

2. In this case we want to show, that we achieve the same petri net by first performing the model transformation and afterwards the operational semantics for petri nets.

   Therefore, we apply the model transformation like before and get as a result the model shown in Figure 65.

   Now, we have to perform the operational semantics in exactly the same order as before. In this case we have to use the target component of the integrated model of the rule *directTransition*, which lets the transition labelled *produce* consume the token on the place *Producing* and put one token on *Wait4cons*.

   Finally, we apply the rule *indirectTransition* once, which removes the token from *Wait4Cons* and puts one token on *Producing*. Like in the state machine before, this is not a valid operation yet and we still need to apply the rule *directTransition*. Doing so, will result in the petri net shown in Figure 67 on the right.

We can conclude, that it is possible in a case study to use the borrowed context for the propagation of operational semantics between state machines and elemental petri nets. In future work, this has to be extended to a more general way, with generalized operational semantics, transformation rules. Furthermore, this has to be examined in a more complex example for a state machine. It is to define a transformation for the case, that a state ,

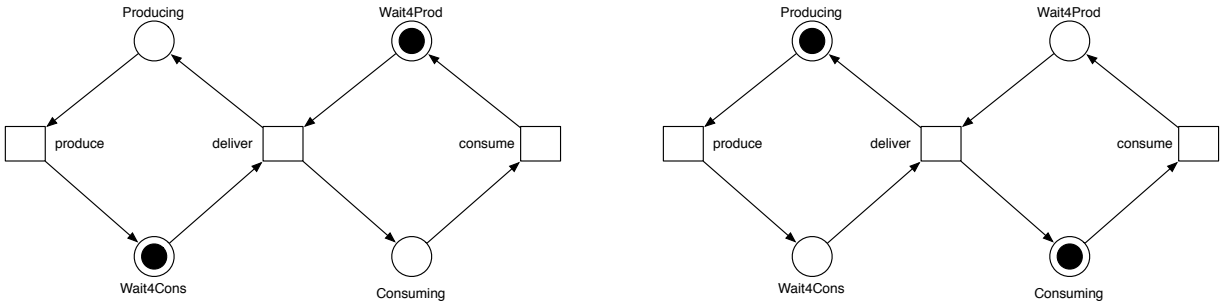


Figure 67: The Producer-Consumer PT net after step 1 and step 2 of OP

that is in the condition of a transition, is not connected to the same event, i.e., there has to be another way on the petri net domain to confirm, that the place has a token.

In general, it needs to be defined, how to transform operational semantics and which conditions need to be satisfied in order to make it applicable in the source and the target domain.

6 Conclusion

In this section we conclude this diploma thesis. In Section *6.1 Summary of Results* we give an overview of the results achieved in this thesis. We discuss what can be done to continue to work with the structures and constructions introduced in the thesis in the Section *6.2 Future Work*.

6.1 Summary of Results

This diploma thesis focuses on model transformation with Triple Graph Grammars [Sch94]. With this technique complete, correct models can be transformed using algebraic transformation [EEPT06]. Since 1994, extensions of the original TGG definitions have been published [KS06] and applications have been presented [KW07].

The first main objective of this thesis was to define a model transformation for model fragments, that is based on algebraic graph transformation (see [EEPT06]). This has been achieved by devoloping the approach of the borrowed context by [EK06] further to algebraic graph transformation (see Section 3). With this approach, graph transformation without a complete match of the rule is possible through extending the graph that we want to transform with information from the rule. On the basis on triple graph transformation (see [Sch94]) we also defined the model transformation with borrowed context in this section. Moreover, we introduced the notion of source consistency [EEHP09] with borrowed context, which allows us to check, if for a forward transformation with borrowed context there exists a corresponding source sequence. Furthermore, we defined a forward consistent BC-match and an on-the-fly construction for borrowed context model transformation sequences was given, such that we have source consistence for our sequence by construction. In Theorem 7

we prooved, that every BC-forward transformation sequence can be extended to a standard transformation sequence, which is essential for the application of this thesis to the standard model transformation with triple graph grammars. In Section 3 we also defined the terms for correct, complete and terminating BC-model transformations.

Our second main objective was to propagate graph constraints with borrowed context model transformations, which was achieved in Section 4. We introduced a method to propagate a graph constraint and afterwards defined under which a model satisfies the propagated constraint.

Finally, in Section 5, we presented a case study in the modelling framework ABT-Reo [BHE09], [BH09], where we reconstructed the theory developed in the previous section in a concrete example. We first transformed a complete model with a standard model transformation. Thereafter, we transformed a graph constraint with borrowed context and showed, that the integrated model after the transformation satisfies the propagated constraint. Moreover, we used the software tool AGG [AGG] for the analyzation of ABT-Reo to show, that it has no critical pairs, which is essential for Theorem 10, which states, that the triple language needs to have strong functional behaviour.

Then we introduced another case study on the transformation of operational semantic rules between State Machines [EEPT06] and Petri Nets [Rei84]. We showed, that it is possible to use borrowed context transformation in order to transform the rules and that the model properties were preserved.

6.2 Future Work

In future, it would be interesting to define the propogation of a graph constraint in a way, such that we get only a constraint in the target domain without the source component in the triple graphs. Then, the soundness of such propagation should be defined, such that we can say, when a model from the target language satisfies such a translated constraint, i.e., $G \models MT_{BC}(PC(a))$.

Another interesting problem in this field of study is the backwards propagation of a graph constraint, i.e., to define, what we can say about the source model and the source constraint from the propagated constraint: $MT(G) \models MT_{BC}(PC(a)) \Rightarrow G \models PC(a)$.

Furthermore, the borrowed context approach could be used for the transformation of other fragments, such as model morphisms and rules. In particular, the propagation of rules for operational semantics seems interesting and to define, under which conditions the properties of the operational semantics is preserved, i.e., first transforming a model and then applying the propagated rules for operational semantics to the transformed model or first applying the rules for operational semantics to the source model and transforming it afterwards are equivalent.

In the case study in Section **??** an outlook was given on transforming rules for operational semantics. This needs to be defined in further detail and furthermore, it would be interesting to find out under which conditions the general preservation of model properties can be ensured when transforming operational semantics, such that the behaviour of the operations in the models is equivalent.

References

[AGG] AGG Homepage. `http://tfs.cs.tu-berlin.de/agg`.

[BH09] Christoph Brandt and Frank Hermann. Modeling and Reconfiguration of critical Business Processes for the purpose of a Business Continuity Management respecting Security, Risk and Compliance requirements at Credit Suisse using Algebraic Graph Transformation: Long Version. Technical report, TU Berlin, Fak. IV, 2009. (to appear).

[BHE09] Christoph Brandt, Frank Hermann, and Thomas Engel. Security and Consistency of IT and Business Models at Credit Suisse realized by Graph Constraints, Transformation and Integration using Algebraic Graph Theory. In *Proc. Int. Conf. on Exploring Modeling Methods in Systems Analysis and Design 2009 (EMMSAD'09)*, volume 29 of *LNBIP*, pages 339–352, Heidelberg, 2009. Springer Verlag.

[EEE+07] Hartmut Ehrig, Karsten Ehrig, Claudia Ermel, Frank Hermann, and Gabriele Taentzer. Information preserving bidirectional model transformations. In Matthew B. Dwyer and Antónia Lopes, editors, *Fundamental Approaches to Software Engineering*, volume 4422 of *LNCS*, pages 72–86. Springer, 2007.

[EEH08] H. Ehrig, K. Ehrig, and F. Hermann. From Model Transformation to Model Integration based on the Algebraic Approach to Triple Graph Grammars. In J. de Lara, C. Ermel, and R. Heckel, editors, *Workshop on Graph Transformation and Visual Modelling Techniques (GT-VMT'08)*, 2008. accepted for publication.

[EEHP09] Hartmut Ehrig, Claudia Ermel, Frank Hermann, and Ulrike Prange. On-the-Fly Construction, Correctness and Completeness of Model Transformations based on Triple Graph Grammars: Long Version. Technical report, Technische Universität Berlin, 2009. Available online at `http://tfs.cs.tuberlin.de/publikationen/Papers08/EEHP09.pdf`.

[EEPT06] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of Algebraic Graph Transformation (Monographs in Theoretical Computer Science. An EATCS Series)*. Springer, 1 edition, 3 2006.

[Ehr79] H. Ehrig. Introduction to the Algebraic Theory of Graph Grammars (A Survey). In *Graph Grammars and their Application to Computer Science and Biology*, volume 73, pages 1–69. 1979.

[EK06] Hartmut Ehrig and Barbara König. Deriving bisimulation congruences in the dpo approach to graph rewriting with borrowed contexts. *Mathematical Structures in Computer Science*, 16(6):1133–1163, 2006.

[HEGO10] F. Hermann, H. Ehrig, U. Golas, and F. Orejas. Efficient analysis and execution of correct and complete model transformations based on triple graph grammars. In *Proceedings of the Workshop on Model Driven Interoperability (MDI'2010)*, 2010. to appear.

[HEOG10] F. Hermann, H. Ehrig, F. Orejas, and U. Golas. Formal analysis of functional behaviour for model transformations based on triple graph grammars. In *Proceedings of Intern. Conf. on Graph Transformation (ICGT' 10)*, 2010. to appear.

[Her08] Frank Hermann. Process Definition of Adhesive HLR Systems. Forschungsberichte der Fakultät IV 2008/09, TU Berlin, Fak. IV, Faculty IV of TU Berlin, Berlin, Germany, 2008. (to appear).

[KS06] Alexander Königs and Andy Schürr. Tool integration with triple graph grammars - a survey. *Electr. Notes Theor. Comput. Sci.*, 148(1):113–150, 2006.

[KW07] E. Kindler and R. Wagner. Triple graph grammars: Concepts, extensions, implementations, and application scenarios. Technical Report TR-ri-07-284, Department of Computer Science, University of Paderborn, Germany, 2007.

[LS04] Stephen Lack and Pawel Sobocinski. Adhesive categories. In *FoSSaCS*, pages 273–288, 2004.

[Rei84] Wolfgang Reisig. *On the Semantics of Petri Nets.* Univ. Hamburg, Fachbereich Informatik, Bericht Nr. 100, September 1984.

[Sch94] A. Schürr. Specification of Graph Translators with Triple Graph Grammars. In G. Tinhofer, editor, *WG94 20th Int. Workshop on Graph-Theoretic Concepts in Computer Science*, volume 903 of *Lecture Notes in Computer Science*, pages 151–163, Heidelberg, 1994. Springer Verlag.