Software and Systems Modeling, submission, manuscript No.
(will be inserted by the editor)

Model Synchronization Based on Triple Graph Grammars

Correctness, Completeness and Invertibility

Frank Hermann[1], Hartmut Ehrig[2], Fernando Orejas[3], Krzysztof Czarnecki[4], Zinovy Diskin[4], Yingfei Xiong[5,6], Susann Gottmann[1], Thomas Engel[1]

[1] Interdisciplinary Center for Security, Reliability and Trust, Université du Luxembourg,
e-mail: {frank.hermann,susann.gottmann,thomas.engel}(at)uni.lu
[2] Institut für Softwaretechnik und Theoretische Informatik, Technische Universität Berlin, Germany,
e-mail: ehrig(at)cs.tu-berlin.de
[3] Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, Barcelona, Spain,
e-mail: orejas(at)lsi.upc.edu
[4] Generative Software Development Lab, University of Waterloo, Canada
e-mail: {kczarnec, zdiskin}@gsd.uwaterloo.ca
[5] Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, China
e-mail: xiongyf@pku.edu.cn
[6] Institute of Software, School of Electronics Engineering and Computer Science, Peking University, China

Received: 19-APRIL-2012 / Revised version: 31-OCTOBER-2012

Abstract Triple graph grammars (TGGs) have been used successfully to analyze correctness and completeness of bidirectional model transformations, but a corresponding formal approach to model synchronization has been missing. This paper closes this gap by providing a formal synchronization framework with bidirectional update propagation operations. They are generated from a given TGG, which specifies the language of all consistently integrated source and target models.

As our main result, we show that the generated synchronization framework is correct and complete, provided that forward and backward propagation operations are deterministic. Correctness essentially means that the propagation operations preserve and establish consistency while completeness ensures that the operations are defined for all possible inputs. Moreover, we analyze the conditions under which the operations are inverse to each other. All constructions and results are motivated and explained by a running example, which leads to a case study, using concrete visual syntax and abstract syntax notation based on typed attributed graphs.

Keywords: model synchronization, correctness, bidirectional model transformation, triple graph grammars

1 Introduction

Bidirectional model transformations are a key concept for model generation and synchronization within model driven engineering (MDE, see [36,31,5]). Triple graph grammars (TGGs) have been successfully applied in several case studies for bidirectional model transformation, model integration and synchronization [29,35,15,14], and in the implementation of QVT [19]. Inspired by Schürr et al. [33,35], we started to develop a formal theory of TGGs [12,22], which allows us to handle correctness, completeness, termination, and functional behavior of model transformations.

The main goal of this article is to provide a TGG framework for model synchronization with correctness guarantees, which is based on the theory of TGGs, work on incremental synchronization by Giese et al. [15,14], and the model synchronization framework [8]. The main ideas and results are the following:

1. Models are synchronized by propagating changes from a source model to a corresponding target model using forward and backward propagation operations. The operations are specified by a TGG model framework, inspired by symmetric replica synchronizers [8] and realized by model transformations based on TGGs [12]. The specified TGG also defines consistency of source and target models.
2. Since TGGs define, in general, non-deterministic model transformations, the derived synchronization operations are, in general, non-deterministic. But we are able to provide sufficient static conditions based on TGGs to ensure that the operations are deterministic.
3. The main result shows that a TGG synchronization framework with deterministic synchronization operations is correct, i.e., consistency preserving, and complete. We also give sufficient static conditions for invertibility and

Correspondence to: frank.hermann(at)uni.lu, xiongyf@pku.edu.cn

weak invertibility of the framework, where "weak" restricts invertibility to a subclass of inputs.

Deriving a synchronization framework from a TGG has the following practical benefits. Consistency of the related domains is defined declaratively and in a pattern-based style, using the rules of a TGG. After executing a synchronization operation, consistency of source and target models is always ensured (correctness) and the propagation operations can be performed for all valid inputs (completeness). The required static conditions of a TGG and the additional conditions for invertibility can be checked automatically using the existing tool support of AGG [37].

This article is based on previous work in [23] and provides extended explanations and results. In particular, we present the technical details concerning determinism of TGG operations in Sec. 6 including the corresponding formal result in Thm. 8.1. Concerning correctness, completeness and invertibility of the synchronization framework, we provide extensive technical details leading to the two main theorems Thm. 8.2 and Thm. 8.6. Moreover, we present the complete TGG of our case study throughout the paper including the derived operational rules and the detailed analysis results using the tool AGG, while only a small subset of the rules is presented in [23].

The next section presents our running example and a general motivation. Sec. 3 introduces the TGG model synchronization framework and Sec. 4 reviews model transformations based on TGGs. Thereafter, Sec. 5 provides extended concepts concerning efficient executions based on marking rules. We present the automated analysis techniques for ensuring deterministic behavior in Sec. 6. On this basis, we describe and illustrate the general synchronization process in Sec. 7. Thereafter, Sec. 8 presents the main results on correctness, completeness and invertibility of the model synchronization framework. Finally, Sec. 9 and 10 discuss related work, conclusions, and future work. The proofs of technical results are provided in a technical report [24].

2 Example and Motivation

Throughout the paper, we use a simple running example, which is based on previous work [6]. The example considers the synchronization of two organizational diagrams as shown in Fig. 2.1. Diagrams in the first domain — depicted left — provide a view on employees of the marketing department of a company, while diagrams in the second domain — depicted right — show all employees. Furthermore, both domains differ on the type of information they specify for a person. Diagrams on the left show the base and bonus salary values of each person, while diagrams in the second domain show only the total salary for each person, but additionally, they provide the birth dates (marked by "*") for each person. Therefore, both domains contain exclusive information and none of them can be interpreted as a view—defined by a query—of the other. Both diagrams together with some correspondence structure build up an integrated model, where we refer to the first diagram as the source model and to the second diagram as the target model. Such an integrated model is called *consistent*, if the diagrams coincide on names of corresponding persons, the salary values are equal to the sums of the corresponding base and bonus values, and persons in the source domain are exactly those who are marked with "M" in the target domain.

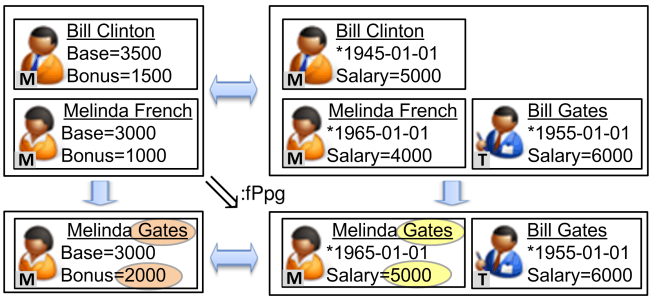

Fig. 2.1 Forward propagation

Example 2.1 (Update Propagation) The first row of Fig. 2.1 shows a consistent integrated model M in a visual notation. The source model of M consists of two persons belonging to the marketing department (depicted as persons with label "M" and without pencils) and the target model additionally contains the person "Bill Gates" belonging to the technical department (depicted as a person with label "T" and with pencil). The first column shows an update of the source model, where person "Bill Clinton" is removed and some attribute values of person "Melinda French" are modified. This change is propagated to the target domain leading to a target update (right column) and a new integrated model (bottom row).

The synchronization problem is to propagate a model update in a way, such that the resulting integrated model is consistent. Looking at Fig. 2.1, this requires that the source model update of removing person "Bill Clinton" and changing the attributes LastName and Bonus of person "Melinda French" is propagated in an appropriate way to the target domain. In this example, this means that the executed forward propagation (fPpg) shall remove person "Bill Clinton" and update the attribute values of "Melinda French" in the target model, such that the unchanged birth date value and consistency is preserved.

Synchronization scenarios like the one in our example, are present in many domains. Consider for example synchronizations between different kinds of visual models for software development, models for software analysis and even source code. Synchronizations between these domains often need to provide mechanisms that do not require that one model can be completely obtained from the other. In other words, none of the models is just a view of the other. In this article, we will show how this flexibility in the synchronization process is possible based on the formal notion of triple graph grammars. We stepwise develop the required formal techniques and illustrate them on the running example in

Fig. 2.1, whose intermediate steps are presented in Fig. 7.2 of Sec. 7.

3 Model Synchronization Framework

In this section, we describe the basic framework that we consider in the paper. First, we introduce the notion of *triple graphs* to describe pairs of interrelated models and *triple graph grammars* (TGGs) as a tool to specify classes of what we consider consistent interrelated models [33]. Then we define the notion of a *TGG model framework* to describe the basic TGG setting that we must consider to define and solve synchronization problems. The framework is a simplified version of the symmetric delta lens proposed by Diskin et al. [8]. Finally, we define formally *synchronization problems*, and the *propagation operations* that are needed to solve them. In particular, a model framework becomes a *synchronization framework* when these operations are considered. Moreover, we state some properties that in our view propagation operations must satisfy.

Model synchronization aims to achieve consistency among interrelated models. In particular, we consider that a model is some kind of graph and that graphs are related by means of graph morphisms consisting of two functions that map nodes to nodes and edges to edges, respectively, in a consistent way. Moreover, we consider that a pair of interrelated models (M^S, M^T), called *source and target models*, are represented by a *triple graph*, which we also call an *integrated model*. This triple graph consists of three graphs G^S (representing M^S), G^C, and G^T (representing M^T), called source, correspondence, and target graphs, respectively, together with two mappings (graph morphisms) $s_G : G^C \to G^S$ and $t_G : G^C \to G^T$. These two mappings specify a *correspondence* $r : G^S \leftrightarrow G^T$, which relates the elements of G^S with their corresponding elements of G^T and vice versa. In addition, our triple graphs may also contain attributed nodes and edges [12, 11].

For simplicity, we use double arrows ($\leftrightarrow$) as an equivalent shorter notation for triple graphs, whenever the explicit correspondence graph can be omitted.

A triple graph morphisms $m : G \to H$ relates two triple graphs G and H. It consists of three graph morphisms that preserve the associated correspondences (i.e., the diagrams in Fig. 3.1 commute).

$$\begin{array}{ccccc} G = (G^S & \xleftarrow{s_G} & G^C & \xrightarrow{t_G} & G^T) \\ {\scriptstyle m}\downarrow & {\scriptstyle m^S}\downarrow & & {\scriptstyle m^C}\downarrow & {\scriptstyle m^T}\downarrow \\ H = (H^S & \xleftarrow[s_H]{} & H^C & \xrightarrow[t_H]{} & H^T) \end{array}$$

Fig. 3.1 Triple graph morphism

Our graphs and triple graphs are typed. This means that a type triple graph TG is given (playing the role of a metamodel) and, moreover, every triple graph G is typed by a triple graph morphism $type_G : G \to TG$. It is required that morphisms between typed triple graphs preserve the typing. Triple graphs specify the abstract syntax of visual languages in an integrated way. For $TG = (TG^S \leftarrow TG^C \to TG^T)$, we use $VL(TG)$ (integrated models), $VL(TG^S)$ (source domain), and $VL(TG^T)$ (target domain) to denote the classes of all graphs typed over TG, TG^S, and TG^T, respectively.

A TGG specifies a language of triple graphs, which are considered as consistent integrated models. The triple rules of a TGG are used to synchronously build up source and target models, together with the correspondence structures.

$$\begin{array}{ccccc} L = (L^S & \xleftarrow{s_L} & L^C & \xrightarrow{t_L} & L^T) \\ {\scriptstyle tr}\downarrow & {\scriptstyle tr^S}\downarrow & & {\scriptstyle tr^C}\downarrow & {\scriptstyle tr^T}\downarrow \\ R = (R^S & \xleftarrow[s_R]{} & R^C & \xrightarrow[t_R]{} & R^T) \end{array} \qquad \begin{array}{ccc} L & \xrightarrow{tr} & R \\ {\scriptstyle m}\downarrow & (PO) & \downarrow{\scriptstyle n} \\ G & \xrightarrow[t]{} & H \end{array}$$

Fig. 3.2 Triple rule and triple graph transformation step

A triple rule tr, as depicted in Fig. 3.2, is an inclusion of triple graphs, represented $L \hookrightarrow R$. It specifies how a given consistent integrated model can be extended simultaneously on all three components yielding again a consistent integrated model. In particular, this means that triple rules are non-deleting. This is sufficient, because triple rules are not used for editing in the source and target domains. Moreover, as shown in [22], triple rules can be extended by negative application conditions (NACs) for restricting their application to specific matches. Notice that one or more of the rule components tr^S, tr^C, and tr^T may be empty. In the example, this is the case for a rule concerning employees of the technical department within the target model. A triple rule is applied to a triple graph G by matching L to some part of triple graph G. Technically, a match is a morphism $m : L \to G$. The result of this application is the triple graph H, where L is replaced by R in G. Technically, the result of the transformation is defined by a pushout diagram [11] (PO), as depicted in Fig. 3.2 on the right. This triple graph transformation (TGT) step is denoted by $G \overset{tr,m}{\Longrightarrow} H$. From the application point of view, we consider that matches should be injective on the structural part. This means that two distinct nodes (edges) in the left hand side of a rule are never mapped to the same node (edge) in the current triple graph G, i.e., identification of elements in L via a match to G is not possible. But it would be too restrictive to require injectivity of the matches also on the data and variable nodes, because we must allow that two different variables are mapped to the same data value. For this reason we use the notion of almost injective matches, which requires that matches are injective except for the data value nodes. This way, attribute values can still be specified as terms within a rule and matched non-injectively to the same value. For the rest of this paper we generally require almost injective matching for the transformation sequences.

A triple graph grammar $TGG = (TG, S, TR)$ consists of a triple type graph TG, a triple start graph S (typically, the empty triple graph) and a set TR of triple rules. The language generated by TGG, denoted $VL(TGG)$, is the set of all (well-typed) triple graphs that can be generated from the start graph S using the rules in TR. Notice that, as a consequence, $VL(TGG) \subseteq VL(TG)$.

The triple graph grammar $TGG = (TG, \emptyset, TR)$ of our case study is given by the triple type graph TG in Fig. 3.3, the empty start graph and the triple rules in Fig. 3.4.

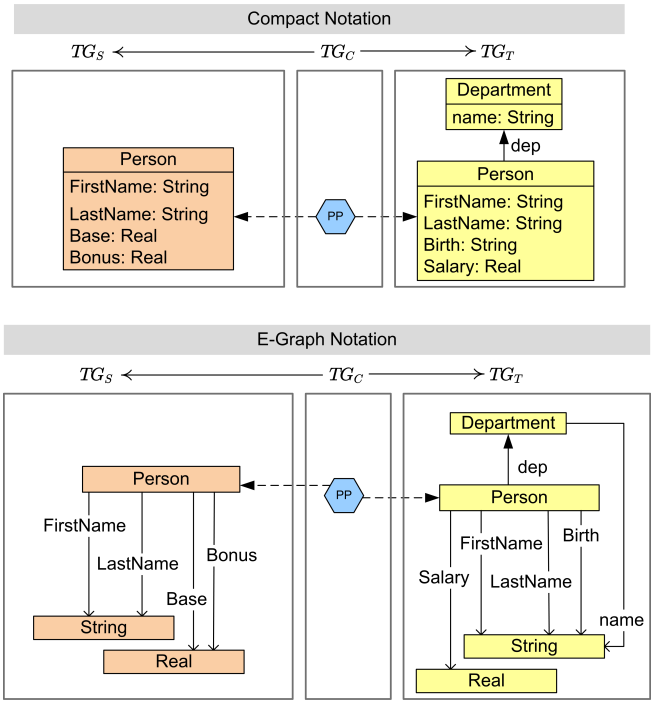

Fig. 3.3 Triple type graph TG

Example 3.1 (Type Graph) The triple type graph TG of our example is shown in Fig. 3.3. It specifies that models of the source domain contain persons including their detailed salary information (bonus and base salary) and their names. Models of the target domain additionally contain the departments to which a person is assigned to, the birth date of a person, and a single value for the complete salary of a person, while the details about bonus and base salary are not provided.

Example 3.2 (Triple Rules) The triple rules of the TGG are depicted in a compact notation in Fig. 3.4. Left- and right-hand side of a rule are depicted in one triple graph and the elements to be created—and, thus, exist in the right-hand side only—have the label "++". The first rule (Person2FirstMarketingP) inserts a new department with name "Marketing" and the NAC ensures that none of the existing departments is named equally. The rule creates a person of the new department in the target model as well as a corresponding person in the source model. Note that the left hand side of this rule is empty, i.e., it does not require existing structures. Rule Person2NextMarketingP is used to extend both models with further persons in the marketing department. The left hand side of this rule contains the department node with name "Marketing". Note that the attributes of the created persons are not set with these rules. This is possible in our formal framework of attributed graph transformation based on the notion of E-graphs [11]. The main advantage is

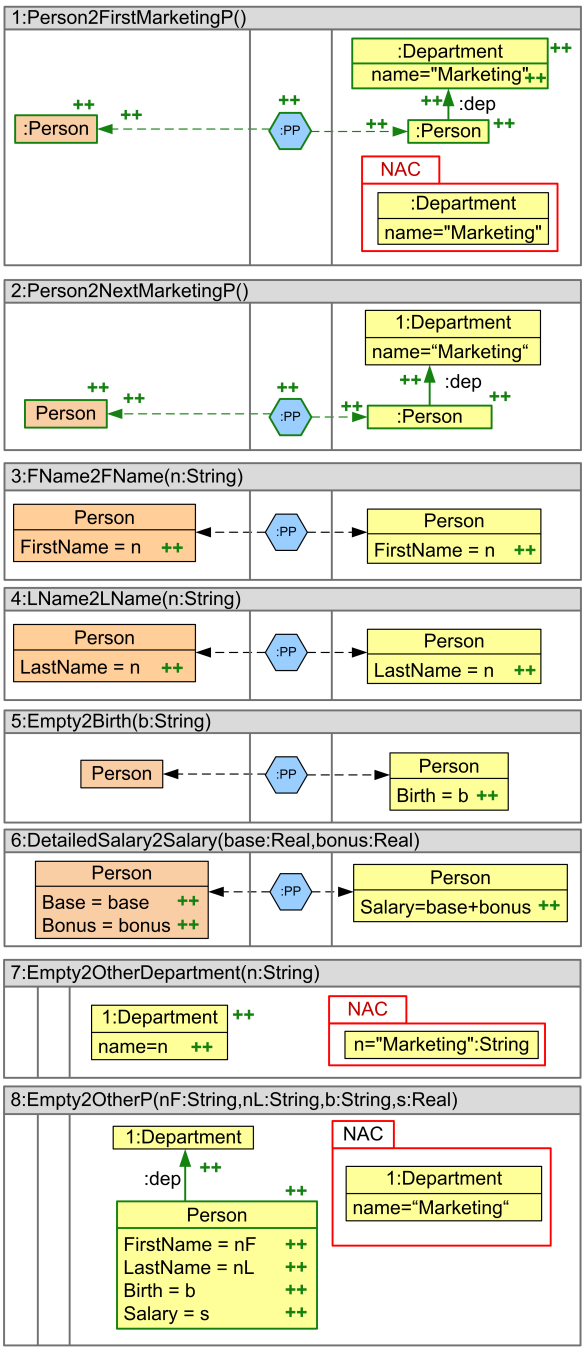

Fig. 3.4 Triple rules

that we can propagate changes of attribute values without the need for deleting and recreating the owning structural nodes. This is important from the efficiency and application point of view. Thus, rules 3-6 concern the creation of attribute values only. Rules 3 (FName2FName) and 4 (LName2LName) create new corresponding values for first and last names, respectively. The next rule (Empty2Birth) assigns the birth date of a person in the target component and does not change the source component. Finally, rule 6 (DetailedSalary2Salary) assigns the detailed salary values (bonus and base) in the source component and the sum of them in the target component. Rule 7 (Empty2OtherDepartment) creates a new de-

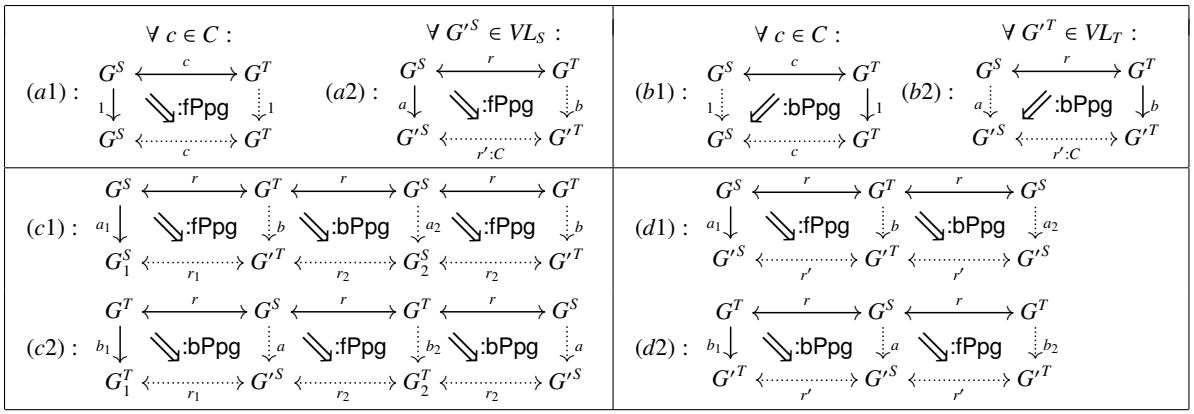

Fig. 3.5 Laws for correct and (weak) invertible synchronization frameworks

partment that is not named "Marketing", but does not change the source model. The negative application condition (NAC) ensures that the used attribute value is different from "Marketing". The last rule Empty2OtherP of the TGG creates a new person of a department that is not in the marketing department. Therefore, there are no correspondences to the source model and the rule directly creates the person including all attribute values.

A *TGG model framework* specifies the possible correspondences between models and updates of models for a given TGG, according to Def. 3.3 below. More precisely, a model framework is defined as consisting of the classes of well-typed source and target models, the class of correspondences between source and target models (i.e., the class of well-typed triple graphs), the subset of consistent correspondences (i.e., the class of triple graphs defined by the given TGG) and the classes of source and target updates. In particular, a model update $\delta : G \to G'$ is specified as a *graph modification* consisting of two inclusions, $\delta : G \hookleftarrow I \hookrightarrow G'$. The intuition of a graph modification is that the inclusion $I \hookrightarrow G$ specifies the elements that are deleted from G (all the elements that are not in I) and $I \hookrightarrow G'$ specifies all the elements that are added by δ (all the elements in G' that are not in I). Therefore, the elements in I are the elements that remain invariant after the modification. It may be noted that graph modifications look like triple graphs, however their role is different: triple graphs are used to make explicit the interrelations between two integrated models, while graph modifications are used to describe updates on a given model.

Definition 3.3 (TGG Model Framework) *Let TGG* $= (TG, \varnothing, TR)$ *be a triple graph grammar with empty start graph* $\varnothing$ *and triple type graph TG containing source and target components* TG^S *and* TG^T*, and a set TR of triple rules. The derived* TGG model framework $MF(TGG) = (VL(TG^S), VL(TG^T), R, C, \Delta_S, \Delta_T)$ *consists of source domain* $VL(TG^S)$*, target domain* $VL(TG^T)$*, the set R of correspondence relations given by* $R = VL(TG)$*, the set C of consistent correspondence relations* $C \subseteq R$ *given by* $C = VL(TGG)$*, (i.e., R contains all integrated models and C all consistent integrated ones), and sets* Δ_S, Δ_T *of graph modifications for the source and target domains, given by* $\Delta_S = \{a : G^S \to G'^S \mid G^S, G'^S \in VL(TG^S)$*, and a is a graph modification*$\}$ *and* $\Delta_T = \{b : G^T \to G'^T \mid G^T, G'^T \in VL(TG^T)$*, and b is a graph modification*$\}$*, respectively.*

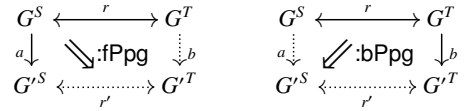

Fig. 3.6 Synchronization operations

Given a TGG model framework, *the synchronization problem* is to provide suitable total and deterministic forward and backward operations fPpg and bPpg that propagate updates on one model (G^S or G^T) to the other model. More precisely, given an integrated model (a correspondence relation) $G^S \leftrightarrow G^T$ and an update $a : G^S \to G'^S$, the operation fPpg must propagate the update a to G^T returning as results an update $b : G^T \to G'^T$ and a correspondence relation $G'^S \leftrightarrow G'^T$. Similarly, bPpg is the dual operation that propagates updates on target models to updates on source models. The effect of these operations is depicted schematically in the diagrams on Fig. 3.6, which we call *synchronization tiles*, where we use solid lines for the inputs and dashed lines for the outputs [8]. Note that, in a common tool environment, the required input for these operations is either available directly or can be obtained. For example, the graph modification of a model update can be derived via standard difference computation and the initial correspondence can be computed based on TGG integration concepts [10,29]. Note also that determinism of fPpg means that the resulting correspondence $G'^S \leftrightarrow G'^T$ and update $b : G^T \to G'^T$ are uniquely determined. The propagation operations are *correct*, if they additionally preserve consistency as specified by laws $(a1) - (b2)$ in Fig. 3.5. Law $(a2)$ means that fPpg always produces consistent correspondences from consistent updated source models G'^S. Law $(a1)$ means that if the given update is the identity and the given correspondence is consistent, then fPpg changes nothing. Laws $(b1)$ and $(b2)$ are the dual versions

concerning bPpg. Moreover, the sets VL_S and VL_T specify the *consistent source and target models*, which are given by the source and target components of the integrated models in $C = VL(TGG)$.

Definition 3.4 (Synchronization Problem and Framework) *Let* $MF = (VL(TG^S), VL(TG^T), R, C, \Delta_S, \Delta_T)$ *be a TGG model framework. The* forward synchronization problem *is to construct an operation* fPpg $: R \otimes \Delta_S \to R \times \Delta_T$ *leading to the left diagram in Fig. 3.6, where* $R \otimes \Delta_S = \{(r,a) \in R \times \Delta_S \mid r\colon G^S \leftrightarrow G^T, a\colon G^S \to G'^S\}$*, i.e., a and r coincide on* G^S*. The pair* $(r,a) \in R \otimes \Delta_S$ *is called premise and* $(r',b) \in R \times \Delta_T$ *is called solution of the forward synchronization problem, written* $\mathsf{fPpg}(r,a) = (r',b)$*. The* backward synchronization problem *is to construct an operation* bPpg *leading to the right diagram in Fig. 3.6. Operation* fPpg *is called* correct *with respect to C, if axioms* $(a1)$ *and* $(a2)$ *in Fig. 3.5 are satisfied and, symmetrically,* bPpg *is called* correct *with respect to C, if axioms* $(b1)$ *and* $(b2)$ *are satisfied.*

Given total and deterministic propagation operations fPpg *and* bPpg*, the derived* synchronization framework *Synch*(TGG) *is given by Synch*$(TGG) = (MF, \mathsf{fPpg}, \mathsf{bPpg})$*. It is called* correct*, if* fPpg *and* bPpg *are correct; it is* weakly invertible *if axioms* $(c1)$ *and* $(c2)$ *in Fig. 3.5 are satisfied; and it is* invertible *if additionally axioms* $(d1)$ *and* $(d2)$ *in Fig. 3.5 are satisfied.*

Invertibility (laws $(d1)$ and $(d2)$) means that the propagation operations are essentially inverse of each other. For instance, axiom $(d1)$ states that if we propagate update $a_1 : G^S \to G_1^S$ to G^T obtaining as result update b, and now we propagate update b to G^S, we obtain the same result G_1^S. However, notice that we do not require that the resulting update a_2 must coincide with a_1. In particular, it may be possible that the set of elements of G^S that are not modified by a_1 may not coincide with the set of elements that are not modified by a_2, even if they produce the same result G_1^S (see Ex. 3.5 below). However, as we show in Sec. 8, we are able to ensure the more flexible notion of weak invertibility (laws (c_1) and (c_2)) for our example. More precisely, weak invertibility expresses that the two operations are the inverse of each other, up to certain information that may be lost when applying the operations. For instance, in axiom (c_1) the intuition is that update b, the result of propagation of update a_1, may ignore part of the information added by a_1, because this kind of information may not be relevant for target models. As a consequence, when propagating b to G^S this information would be lost. However, this law also states that no information added by update b would be ignored when propagating it to G^S. The reason is that all that information was, in some sense, included in update a_1, so it must be relevant for source models.

Example 3.5 (Invertibility and Weak Invertibility) Consider a model update b_1 of a given target model, as depicted in Fig. 3.7, where a new person is added together with his birth date, leading to a target model G'^T. The propagation via bPpg

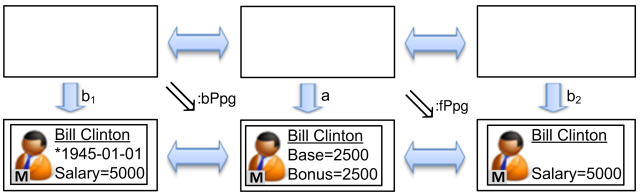

Fig. 3.7 Counter example for invertibility

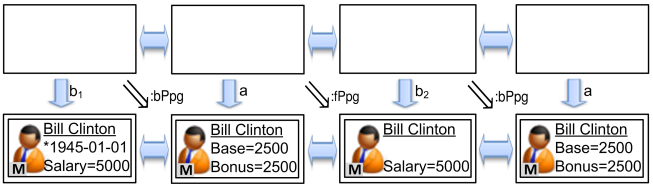

Fig. 3.8 Example for weak invertibility

yields an update a, whose resulting source model G'^S includes that person without his birth date. Now, the propagation of a via fPpg yields an update b_2 whose resulting target model G''^T does not contain any information about the birth date. Therefore, $G'^T \neq G''^T$ meaning that *Synch*(TGG) is not invertible, since law $(d2)$ does not hold. However, if we continue the diagram and perform an additional backward propagation as in Fig. 3.8, we derive a source update that coincides again with a, i.e., the diagrams satisfies law $(c2)$ of weak invertibility.

4 Model Transformation Based on TGGs

In the previous section, we have seen how we can use TGGs to specify the set of consistent correspondences between two classes of models. In this section, we show how we can use TGGs to implement (bidirectional) model transformations [33,12]. More precisely, given a source model G^S (respectively, a target model G^T), the model transformation problem is to find a target model G^T (respectively, a source model G^S) such that $G^S \leftrightarrow G^T$ is a consistent correspondence (or, equivalently, $G^S \leftrightarrow G^T$ belongs to the language generated by the TGG). In particular, we will see that this can be done by means of the *operational transformation rules* that can be generated automatically from a TGG.

There are four classes of operational transformation rules, *source rules* allow us to parse source models, *forward rules* build target models out of source models, *target rules* allow us to parse target models, and finally *backward rules* build source models out of target models. In particular, for a given TGG, the sets TR_S and TR_F including all source and forward rules, respectively, are derived from the triple rules TR in the TGG as shown in Fig. 4.1. Their construction is shown by Ex. 4.1. The sets of target rules TR_T and backward rules TR_B are derived analogously. Moreover, in [12], the generation of operational transformation rules has been extended to triple rules with negative application conditions.

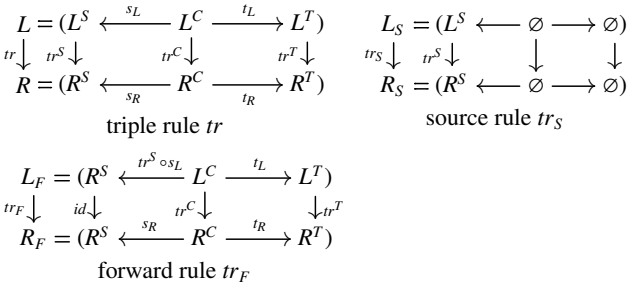

Fig. 4.1 Derived source and forward rules

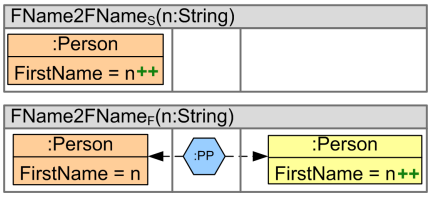

Fig. 4.2 Derived source and forward rules for FName2FName

Example 4.1 (Operational Transformation Rules) The rules in Fig. 4.2 are the derived source and forward rules of the triple rule FName2FName in Fig. 3.4.

The basic idea of how we can build a target model out of a source model using source and forward rules can be explained quite simply[1]. Let us suppose that we want to find a model G^T from a source model G^S such that $G^S \leftrightarrow G^T$ is a consistent correspondence (assuming that this model exists). If $G^S \leftrightarrow G^T$ is consistent this means that, starting from the empty triple graph, there must be a derivation $(\emptyset \leftrightarrow \emptyset) \overset{tr_1}{\Longrightarrow} (G_1^S \leftrightarrow G_1^T) \overset{tr_2}{\Longrightarrow} \ldots \overset{tr_n}{\Longrightarrow} (G_n^S \leftrightarrow G_n^T)$, where $G_n^S \leftrightarrow G_n^T = G^S \leftrightarrow G^T$) and $tr_1, \ldots, tr_n$ are triple rules in the TGG. Now we may notice that each triple rule can be seen as the composition of its associated source rule followed by its associated forward rule. This means that the above derivation is equivalent to the derivation $(\emptyset \leftrightarrow \emptyset) \overset{tr_{S,1}}{\Longrightarrow} (G_1^S \leftrightarrow \emptyset) \overset{tr_{F,1}}{\Longrightarrow} (G_1^S \leftrightarrow G_1^T) \ldots \overset{tr_{S,n}}{\Longrightarrow} (G^S \leftrightarrow G_{n-1}^T) \overset{tr_{F,n}}{\Longrightarrow} (G^S \leftrightarrow G^T)$, where each source rule $tr_{S,i}$ and its associated forward rule $tr_{F,i}$ are applied with a compatible match. But, since the application of a source rule $tr_{S,i}$ is independent of the application of any forward rule $tr_{F,j}$, provided that $i < j$, we have that the previous derivation is equivalent to the derivation $(\emptyset \leftrightarrow \emptyset) \overset{tr_{S,1}}{\Longrightarrow} (G_1^S \leftrightarrow \emptyset) \overset{tr_{S,2}}{\Longrightarrow} \ldots \overset{tr_{S,n}}{\Longrightarrow} (G^S \leftrightarrow \emptyset) \overset{tr_{F,1}}{\Longrightarrow} (G^S \leftrightarrow G_1^T) \overset{tr_{F,2}}{\Longrightarrow} \ldots \overset{tr_{F,n}}{\Longrightarrow} (G^S \leftrightarrow G^T)$, where each source rule $tr_{S,i}$ and its associated forward rule $tr_{F,i}$ are applied with a compatible match.

In this sense, we say that a forward transformation sequence $(G_0 \overset{tr_F^*}{\Longrightarrow} G_n)$ is *source-consistent* if there is a corresponding source sequence $(\emptyset \overset{tr_S^*}{\Longrightarrow} G_0)$, such that the matches of corresponding source and forward steps are compatible; and, as a consequence, we know that a source model G^S can be transformed into a target model G^T if there is a source-consistent forward transformation sequence $(G^S \leftrightarrow \emptyset) \overset{tr_F^*}{\Longrightarrow} (G^S \leftrightarrow G^T)$. The set of all source consistent forward sequences defines a *model transformation based on forward rules* from the source domain $VL(TG^S)$ to the target domain $VL(TG^T)$.

Definition 4.2 (Model Transformation based on Forward Rules) *A* model transformation sequence $(G^S, G_0 \overset{tr_F^*}{\Longrightarrow} G_n, G^T)$ *consists of a source graph* G^S*, a target graph* G^T*, and a source consistent forward TGT-sequence* $G_0 \overset{tr_F^*}{\Longrightarrow} G_n$ *with* $G^S = G_0^S$ *and* $G^T = G_n^T$*. A* model transformation $MT : VL(TG^S) \Rrightarrow VL(TG^T)$ *is defined by all model transformation sequences* $(G^S, G_0 \overset{tr_F^*}{\Longrightarrow} G_n, G^T)$ *with* $G^S \in VL(TG^S)$ *and* $G^T \in VL(TG^T)$*.*

Hence, source consistency is a control condition that has to be used in the construction of forward transformation sequences, in order to implement a model transformation based on TGGs. In principle, the source sequence is obtained a priori by parsing the given source model in order to guide the forward transformation. However, source and forward sequences can be constructed simultaneously and backtracking can be reduced (c.f. Sec. 6) in order to derive efficient executions of model transformations [12, 22].

Model transformations based on forward rules as defined above are always correct and complete [12, 18, 22] in the following sense. *Correctness* means that for each source model G^S that is transformed into a target model G^T there is a model $G = (G^S \leftarrow G^C \rightarrow G^T)$ in the language of consistent integrated models $VL(TGG)$ defined by the TGG. *Completeness* ensures that for each consistent source model there is a forward transformation sequence transforming it into a consistent target model.

5 Model Transformation and Marking using Translation Attributes

According to the results presented in the previous section, in order to build forward transformation sequences to implement model transformations, we need to use source consistency as a separate control condition. In this section, we show how we can automatically integrate the computation of source consistency in the model transformation process. The idea is to use *translation attributes* and *forward translation rules* instead of standard forward rules. In particular, as shown in [22], this allows for the efficient implementation and analysis of model transformations. We also show that the same idea can be used to partially parse a given triple model G, where partial parsing means finding a maximal consistent submodel $G_0 \subseteq G$. For this reason, the rules needed in this case are called *consistency creating rules*. The section is organized as follows, first we describe what translation attributes are. Then we define forward translation rules and consistency creating rules, and how they are used. Finally, we provide some discussion about termination of transformations using these rules.

[1] The backward case is similar.

	main components	new NAC for each $n : L \to N$ of tr
tr_{CC}	$L_{CC} \xleftarrow{l_{CC}} K_{CC} \xrightarrow{r_{CC}} R_{CC}$ $\| \quad \| \quad \|$ $(R \oplus Att^{\mathbf{T}}_{L} \oplus Att^{\mathbf{F}}_{R\setminus L}) \quad (R \oplus Att^{\mathbf{T}}_{L}) \quad (R \oplus Att^{\mathbf{T}}_{L} \oplus Att^{\mathbf{T}}_{R\setminus L})$	$N_{CC} = (L_{CC} +_L N) \oplus Att^{\mathbf{T}}_{N\setminus L}$
tr_{FT}	$L_{FT} \xleftarrow{l_{FT}} K_{FT} \xrightarrow{r_{FT}} R_{FT}$ $\| \quad \| \quad \|$ $(L_F \oplus Att^{\mathbf{T}}_{L_S} \oplus Att^{\mathbf{F}}_{R_S\setminus L_S}) \quad (L_F \oplus Att^{\mathbf{T}}_{L_S}) \quad (R_F \oplus Att^{\mathbf{T}}_{L_S} \oplus Att^{\mathbf{T}}_{R_S\setminus L_S})$	$N_{FT} = (L_{FT} +_L N) \oplus Att^{\mathbf{T}}_{N_S\setminus L_S}$
tr_{BT}	$L_{BT} \xleftarrow{l_{BT}} K_{BT} \xrightarrow{r_{BT}} R_{BT}$ $\| \quad \| \quad \|$ $(L_B \oplus Att^{\mathbf{T}}_{L_T} \oplus Att^{\mathbf{F}}_{R_T\setminus L_T}) \quad (L_B \oplus Att^{\mathbf{T}}_{L_T}) \quad (R_B \oplus Att^{\mathbf{T}}_{L_T} \oplus Att^{\mathbf{T}}_{R_T\setminus L_T})$	$N_{BT} = (L_{BT} +_L N) \oplus Att^{\mathbf{T}}_{N_T\setminus L_T}$

Fig. 5.1 Components of derived operational translation rules

Translation attributes are boolean-valued attributes that we associate to nodes, edges, and also other attributes to denote if that element has been translated or not, in the case of model transformation, or if it has been parsed or not, in case that we are interested in finding a maximal consistent submodel of a given model. In general, we may not want to add translation attributes to the complete graph, but only to a part of it. For instance, in the case of forward transformation we just need to add attributes to the source part. More precisely, given an attributed graph AG and a subset M of its elements (nodes or edges), we call AG' a graph with translation attributes over AG if it extends AG with one Boolean-valued attribute `tr_x` for each element x in M and one Boolean-valued attribute `tr_x_a` for each attribute associated to such an element x in M. The set M of marked elements, together with all these additional translation attributes is denoted by Att_M.

Using the concept of translation attributes we provide extended operational rules, called *operational translation rules*, such that transformations via these rules do not need to be controlled by a separate control condition. There are three sets of operational translation rules that we derive from a given set TR of TGG-triple rules: TR_{FT} (the set of forward translation rules), TR_{BT} (the set of backward translation rules) and TR_{CC} (the set of consistency creating rules).

A forward translation rule tr_{FT}, introduced in [25], extends the forward rule tr_F by additional Boolean valued translation attributes, which are markers for elements in the source model and specify whether the elements have been translated already. Each forward translation rule tr_{FT} turns the markers of the source elements that are translated by this rule from **F** to **T** (i.e., the elements that are created by tr_S). This way, we can ensure that each element in the source graph is not translated twice, but exactly once. The idea of how forward translation rules are used to implement model transformations works as follows. At the beginning, the source model of a model transformation sequence is extended by translation attributes that are all set to "**F**". Then, the application of a forward translation rule sets to "**T**" all the elements that are translated by the rule. Finally, the model transformation is successfully executed if the source model is completely marked with **T**. However, if we arrive to a model which cannot be further transformed and where some of its translation attributes are **F** then we know that the transformation process has failed and we would probably have to backtrack to find a correct transformation. In the examples, we indicate these markers by check marks in the visual notation and by bold font face in the graph representation.

Due to the modification of the translation attributes, the rules are *deleting*, which means that, technically, the rules cannot be denoted by inclusions $L \hookrightarrow R$. As a consequence, from a formal point of view, triple transformations are not defined as a pushout, but in terms of the classical double pushout (DPO) approach [11]. Moreover, according to the theory of graph transformation ([11]), the application of a deleting graph transformation rule must satisfy the so-called gluing condition. However, in the case of the operational triple rules with translation attributes, this is guaranteed. The reason is that forward translation rules are deleting only on attribution edges, where there are no dangling points, and all identification points are preserved for almost injective matches.

Consistency creating rules are used to compute maximal subgraphs G_k of a given triple graph G typed over TG, such that $G_k \in VL(TGG)$. In the special case that $G \in VL(TGG)$, we know that $G_k \cong G$. Each consistency creating rule switches labels from **F** to **T** for those elements that would be created by the corresponding TGG-rule in TR. This means that elements in the left hand side $L_{CC} = R$ are labeled with **T**, if they are also contained in L, and they are labeled with **F** otherwise. Accordingly, all elements in the right hand side R_{CC} are labeled with **T**.

The *operational translation rules* of a TGG are used for the propagation of changes during a synchronization. They consist of the derived forward translation, backward translation and consistency creating rules.

Definition 5.1 (Operational Translation Rules) *Given a triple rule $tr = (L \to R)$ and its derived source rule $tr_S = (L_S \to R_S)$, target rule $tr_T = (L_T \to R_T)$, forward rule $tr_F = (L_F \to R_F)$ and backward rule $tr_B = (L_B \to R_B)$, the*

derived translation rules of tr are given by consistency creating rule $tr_{CC} = (L_{CC} \xleftarrow{l_{CC}} K_{CC} \xrightarrow{r_{CC}} R_{CC})$, forward translation rule $tr_{FT} = (L_{FT} \xleftarrow{l_{FT}} K_{FT} \xrightarrow{r_{FT}} R_{FT})$, *and* backward translation rule $tr_{BT} = (L_{BT} \xleftarrow{l_{BT}} K_{BT} \xrightarrow{r_{BT}} R_{BT})$ *defined in Fig. 5.1 using the notation based on translation attributes. By* TR_{CC}, TR_{FT}, TR_{BT} *we denote the sets of all derived consistency creating, forward translation and backward translation rules, respectively.*

Remark 5.2 (Construction of Operational Rules) Note that in Fig. 5.1 $(B +_A C)$ is the union of B and C with shared A, such that for instance $(L_{FT} +_L N)$ is the union of L_{FT} and N with shared L. Also, $G \oplus Att_M^{\mathbf{T}}$ denotes adding to the graph G translation attributes for all the elements and attributes included in $M \subseteq G$, and moreover all these attributes are set to **T**. Similarly, $G \oplus Att_M^{\mathbf{F}}$ denotes adding to G all these attributes, but this time they are set to **F**.

Remark 5.3 (Interdependencies between Operational Rules) The consistency creating rules (TR_{CC}) are used in Sec. 7 for marking the already consistent parts of a given integrated model in the second sub-phase of the synchronization. The forward and backward translation rules are used for the third sub-phase. This third sub-phase can be interpreted as a completion of the computed sequence of the second sub-phase. We show in Sec. 7 that this continuation is always possible if the sets of operational rules are deterministic, for which we also provide an automated check and analysis. If a TGG does not ensure deterministic sets of operational rules, the computed maximal subgraph via TR_{CC} may be too large to find a corresponding completion via forward (backward) translation rules. In this case, a possible solution would be to perform backtracking for sub-phases 2 and 3 of the synchronization.

Example 5.4 (Derived Sets of Consistency Creating Rules) Figures 5.2-5.3 show the set of the consistency creating rules derived from the triple rules in Ex. 3.2 according to Def. 5.1. They do not modify the structure of a triple graph, but only the translation attributes. They are used for marking consistent substructures of a given triple graph, i.e., of a given integrated model. By applying all derived consistency creating rules as long as possible to a given triple G graph with all translation attributes set to "**F**", a maximal consistent triple graph that is contained in G is computed. Intuitively, for each element $x \in R$ (node, edge, or attribute) of a triple rule $tr = (L \to R)$ a separate translation attribute (`tr` or `tr_`x) is added for the consistency creating rule tr_{CC}. If an element $x \in R$ is preserved by the triple rule tr ($x \in L$), then the consistency creating rule preserves it as well and the translation attribute has value **T**. Otherwise, if $x \in R$ is created by tr ($x \in R \setminus L$), then it becomes a preserved element in the consistency creating rule tr_{CC} and the corresponding translation attribute is changed from **F** to **T**. In visual notation, this means that all plus signs are replaced by additional translation attributes whose values are changed from **F** to **T** and we denote such a modification by [F ⇒ T].

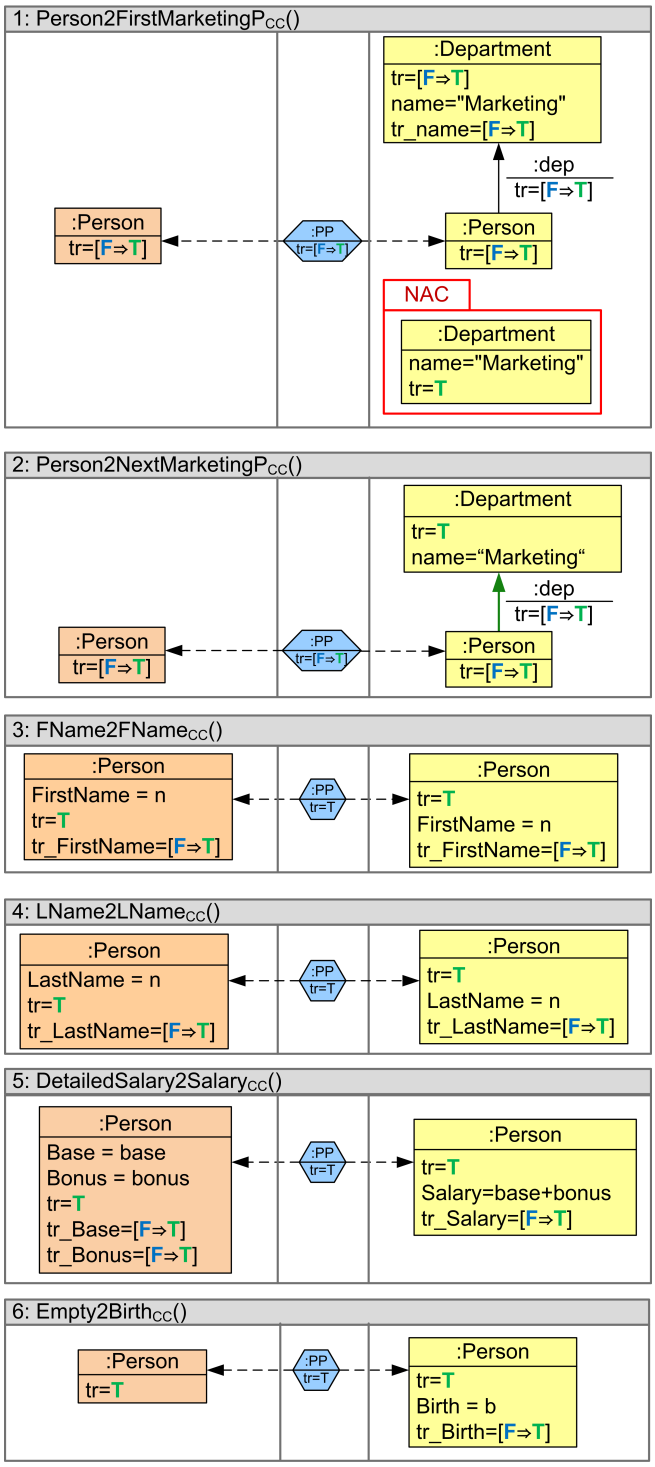

Fig. 5.2 Derived Operational Triple rules: TR_{CC} (part 1)

Example 5.5 (Derived Sets of Forward Translation Rules) Figures 5.4-5.5 show the set of the forward translation rules derived from the triple rules in Ex. 3.2 according to Def. 5.1. These rules are used for translating a source model into its corresponding target model. For this reason, the rules are only modifying the translation attributes on the source component. Intuitively, for each element x in the source component R^S (node, edge, or attribute) of a triple rule $tr = (L \to R)$ a

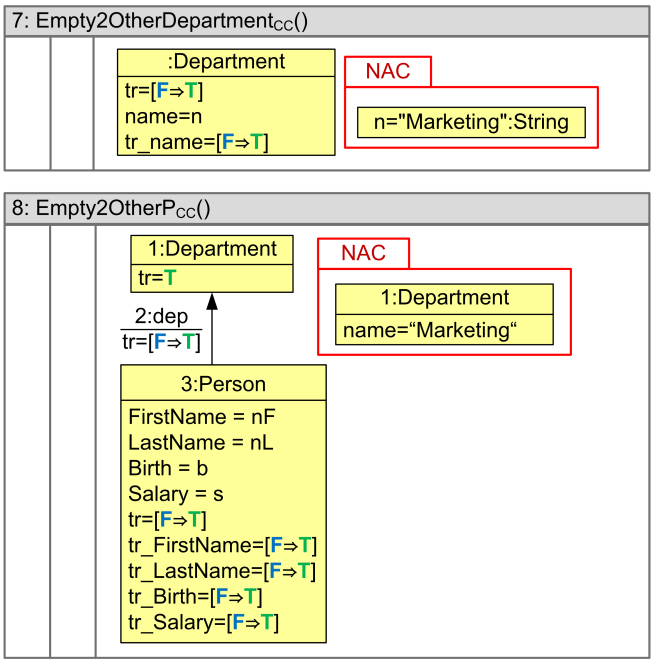

Fig. 5.3 Derived Operational Triple rules: TR_{CC} (part 2)

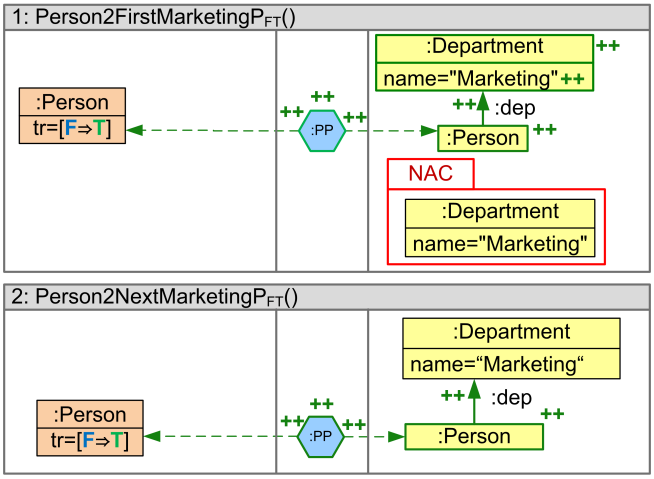

Fig. 5.4 Derived Operational Triple rules: TR_{FT} (part 1)

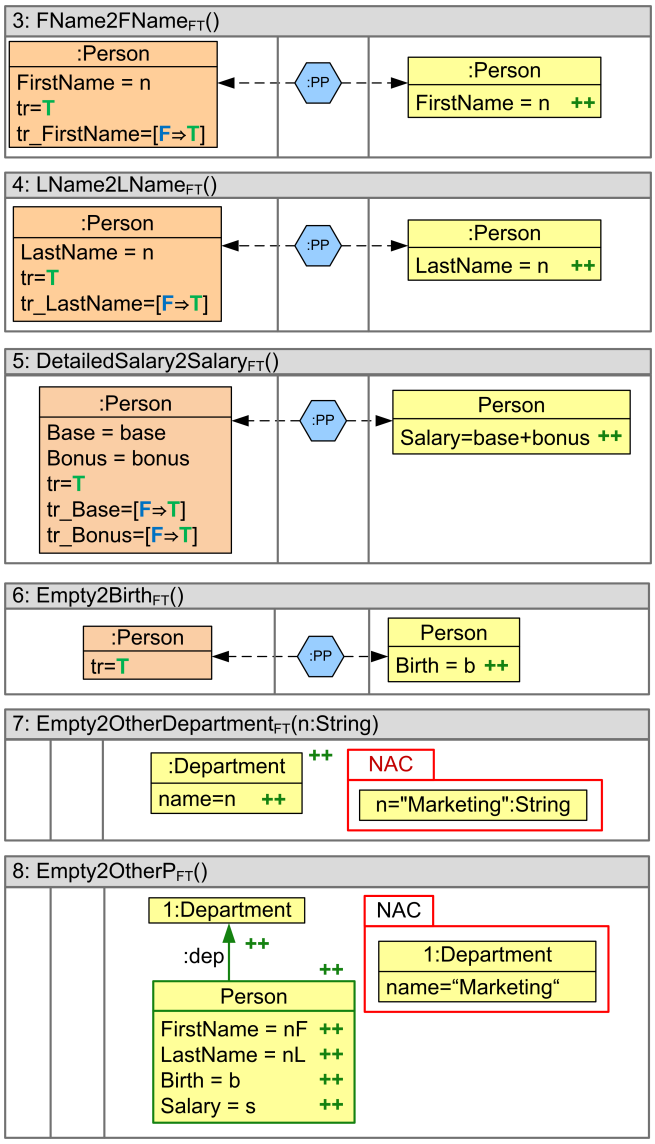

Fig. 5.5 Derived Operational Triple rules: TR_{FT} (part 2)

separate translation attribute (`tr` or `tr_x`) is added for the forward translation rule tr_{FT}. If an element $x \in R^S$ is preserved by the triple rule tr, then the forward translation rule preserves it as well and the translation attribute has value **T**. Otherwise, if $x \in R^S$ is created by tr, then it becomes a preserved element in the forward translation rule tr_{FT} and the corresponding translation attribute is changed from **F** to **T**. In visual notation, this means that each plus sign in the source component of a triple rule is replaced by an additional translation attribute whose value changes from **F** to **T**.

Note that the rules 6-8 are contained in TR^{ls}_{FT}, i.e., they are identities on the source component and according to Def. 3.4, they are not used for fPpg, which is based on TR^{+s}_{FT}. This is important to ensure termination (cf. Rem. 5.10) and we show by Fact 7.8 that the derived sets of operational rules are kernel-grounded (cf. Def. 6.5). For this reason, the reduced set still ensures completeness according to Rem. 6.6 and Thm. 8.2.

Example 5.6 (Derived Sets of Backward Translation Rules) Figures 5.6-5.7 show the set of the backward translation rules derived from the triple rules in Ex. 3.2 according to Def. 5.1. They are derived dually to the case of forward translation rules and used for the translation of target models into their corresponding source models. Thus, they do only modify translation attributes on the target component. Intuitively, for each element x in the target component R^T (node, edge, or attribute) of a triple rule $tr = (L \to R)$ a separate translation attribute (`tr` or `tr_x`) is added for the backward translation rule tr_{BT}. If an element $x \in R^T$ is preserved by the triple rule tr, then the backward translation rule preserves it as well and the translation attribute has value **T**. Otherwise, if $x \in R^T$ is created by tr, then it becomes a preserved element in the backward translation rule tr_{BT} and the corresponding translation attribute is changed from **F** to **T**. In visual notation, this

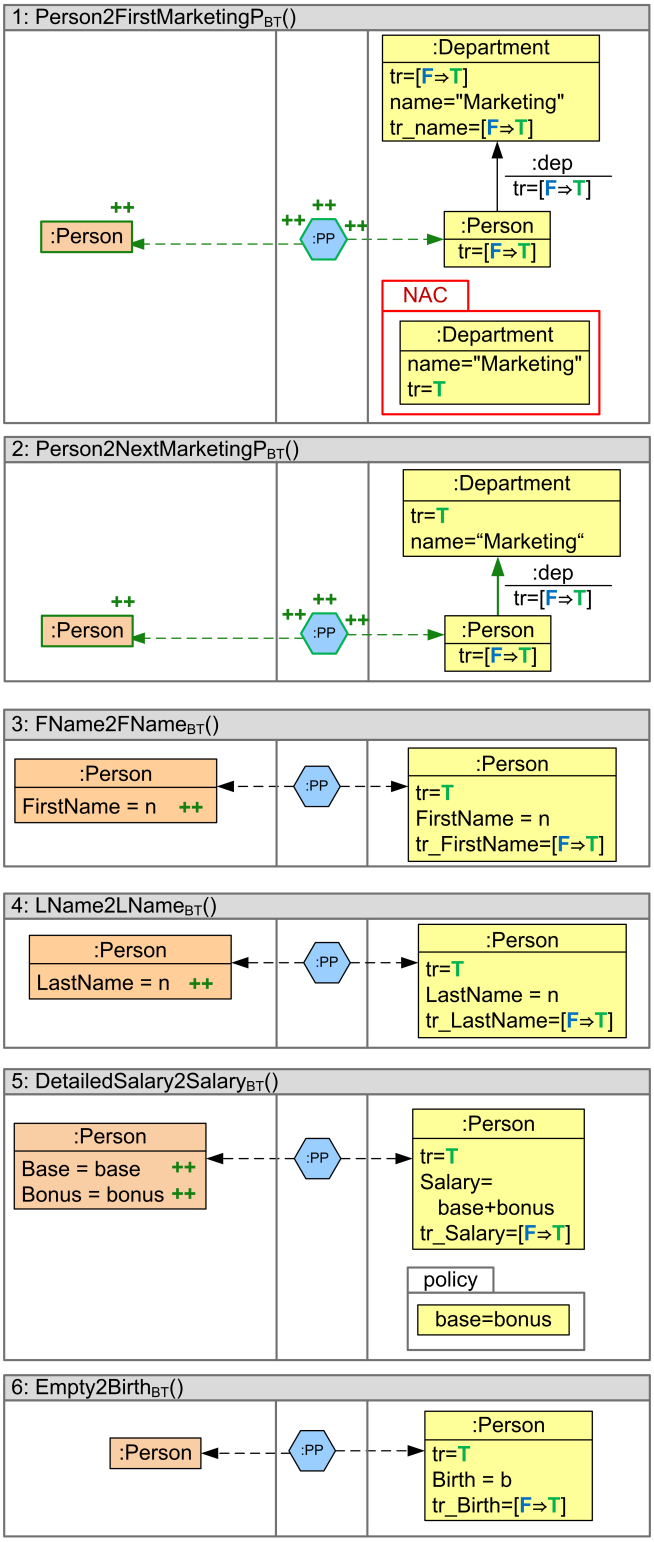

Fig. 5.6 Derived Operational Triple rules: TR_{BT} (part 1)

means that all plus signs in the target component are replaced by additional translation attributes whose values are changed from **F** to **T**. Note that all backward translation rules are used for bPpg in contrast to operation fPpg before.

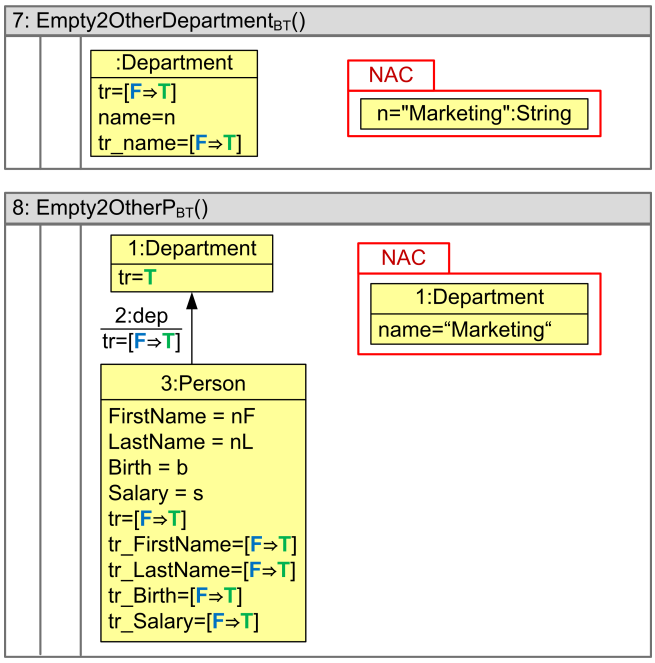

Fig. 5.7 Derived Operational Triple rules: TR_{BT} (part 2)

We define model transformations based on forward translation rules in a similar but slightly different way than for forward rules in Def. 4.2. In both cases we start the transformation process with a triple graph that consists only of the given source graph, i.e., the target and the connection graphs are the empty graphs. But now, the source graph is completely marked with **F**-valued translation attributes, indicating that no element from the graph has been translated yet. Then, instead of applying to the start graph a source consistent forward transformation sequence, we apply a sequence of forward translation transformations leading to a graph whose source part is completely marked with **T**, meaning that all the elements from the given source graph have been translated. These transformation sequences are called *complete forward translation sequences*.

Definition 5.7 (Complete Forward Translation Sequence)
A forward translation sequence $G_0 \overset{tr^*_{FT}}{\Longrightarrow} G_n$ *with almost injective matches is called* complete *if* G_n *is* completely translated*, i.e., all translation attributes of* G_n *are set to true (“***T***”).*

A *model transformation based on forward translation rules* transforms models from the source domain into models of the target domain by executing complete forward translation sequences. Given a concrete source model, then the resulting target model of the model transformation is obtained by restricting the resulting triple graph of the forward translation sequence to the target component. We have shown in [22] that model transformation sequences based on forward rules and those based on forward translation rules, respectively, are equivalent. This ensures that the derived model transformation relations are the same.

Definition 5.8 (Model Transformation Based on Forward Translation Rules) *A* model transformation sequence $(G^S, G'_0 \overset{tr^*_{FT}}{\Longrightarrow} G'_n, G^T)$ *based on forward translation rules* TR_{FT} *consists of a source graph* G^S*, a target graph* G^T*, and a complete TGT-sequence* $G'_0 \overset{tr^*_{FT}}{\Longrightarrow} G'_n$ *typed over* $TG' = TG \oplus Att^{\mathbf{F}}_{|TG^S|} \oplus Att^{\mathbf{T}}_{|TG^S|}$ *based on* TR_{FT} *with* $G'_0 = (Att^{\mathbf{F}}(G^S) \leftarrow \varnothing \rightarrow \varnothing)$ *and* $G'_n = (Att^{\mathbf{T}}(G^S) \leftarrow G^C \rightarrow G^T)$.
A model transformation $MT : VL(TG^S) \Rrightarrow VL(TG^T)$ *based on* TR_{FT} *is defined by all model transformation sequences as above with* $G^S \in VL(TG^S)$ *and* $G^T \in VL(TG^T)$*. All the corresponding pairs* (G^S, G^T) *define the* model transformation relation $MTR_{FT} \subseteq VL(TG^S) \times VL(TG^T)$ *based on* TR_{FT}*. The model transformation is* terminating *if there are no infinite TGT-sequences via* TR_{FT} *starting with* $G'_0 = (Att^{\mathbf{F}}(G^S) \leftarrow \varnothing \rightarrow \varnothing)$ *for some source graph* $G^S \in VL(TG^S)$*.*

Consistency creating sequences as defined in Def. 5.9 below, are used for computing a maximal consistent part of a given triple graph, which is used for the auxiliary operation Del defined in Sec. 7. A consistency creating sequence starts at a triple graph $G'_0 = Att^{\mathbf{F}}(G)$, i.e., at a triple graph where all elements are marked with **F**. Each application of a consistency creating rule modifies some translation attributes of an intermediate triple graph G'_i from **F** to **T** and preserves the structural part G contained in G'_i. Therefore, the resulting triple graph G'_n extends G with translation attributes only, i.e., some are set to **T** and the remaining ones to **F**.

Definition 5.9 (Consistency Creating Sequence) *Given a triple graph grammar* $TGG = (TG, \varnothing, TR)$*, a triple graph* G *typed over* TG *and let* TR_{CC} *be the set of consistency creating rules of* TR*. A* consistency creating sequence $s = (G, G'_0 \overset{tr^*_{CC}}{\Longrightarrow} G'_n, G_n)$ *is given by a TGT sequence* $G'_0 \overset{tr^*_{CC}}{\Longrightarrow} G'_n$ *via* TR_{CC} *with* $G'_0 = Att^{\mathbf{F}}(G)$ *and* $G'_n = G \oplus Att^{\mathbf{T}}_{G_n} \oplus Att^{\mathbf{F}}_{G \setminus G_n}$*, where* G_n *is the subgraph of* G *derived from* $G'_0 \overset{tr^*_{CC}}{\Longrightarrow} G'_n$ *by restricting* G'_n *to all* **T***-marked elements. Consistency creating sequence* s *is called* terminated*, if there is no rule in* TR_{CC} *which is applicable to the result graph* G'_n*. In this case, the triple graph* G'_n *is called a maximal consistency marking of* G*. A triple graph* G' *is called* completely **T**-marked*, if* $G' = Att^{\mathbf{T}}(G)$ *for a given triple graph* G*, i.e., all translation attributes in* G' *are "***T***".*

Remark 5.10 (Termination) It is quite easy to show that, unless the given TGG includes a trivial identical rule $L \hookrightarrow L$, every consistency creating sequence terminates. The reason is that the application of each rule switches some translation predicates from **F** to **T**. Since the number of these predicates in a given triple graph is finite, only a finite number of rule applications is possible.

The case of forward and backward translation sequences is different. In particular, if a triple rule $tr = L \hookrightarrow R$ is *source identic*, meaning that it does not change the source part, i.e., $L^S = R^S$ or equivalently $tr^s = id$, its associated forward translation rule will not switch any translation predicate from **F** to **T**. This implies that this rule could be applied infinitely many often in a forward translation sequence. Something similar happens with backward translation rules.

In this sense, according to whether rules modify or not the source or target part of a rule, we classify rules as shown below. In particular, this notation is used in the following section. Let TR be a set of triple rules. We distinguish the following subsets.

- The set of *source creating* rules $TR^{+s} = \{tr \in TR \mid tr^S \neq id\}$,
- The set of *source identic* rules $TR^{1s} = \{tr \in TR \mid tr^S = id\}$,
- The set of *target creating* rules $TR^{+t} = \{tr \in TR \mid tr^T \neq id\}$,
- The set of *target identic* rules $TR^{1t} = \{tr \in TR \mid tr^T = id\}$, and
- The set of *identic* rules $TR^{1} = \{tr \in TR \mid tr = id\}$.

In order to ensure termination for forward translation sequences, if the given TGG includes source identic triple rules, we propose a general strategy based on an automated analysis using the tool AGG. The main idea is the following. If we can show that none of the remaining triple rules depends on the source identic triple rules, we can actually omit the source identic ones. The reason is that for each forward transformation sequence, we can shift the steps along source identic rules to the end and obtain an equivalent sequence. Since all steps along source identic triple rules do not change the marking of the source model, we further derive that these steps can be removed yielding still a complete forward translation sequence.

6 Deterministic TGGs

Since transformation systems are not deterministic in general, we introduce the concept of policies for transformation rules in order to obtain deterministic sets of operational translation rules for the synchronization operations. The main idea is to restrict the matches of a transformation rule using additional attribute conditions in order to eliminate ambiguous results.

An attribute condition *attCon* for a (triple) rule $tr : L \rightarrow R$ is a set of equations for attribute values. A match $m : L \rightarrow G$ satisfies *attCon*—written $m \models attCon$—if the evaluation of attribute values satisfies each equation. In our case study, we use one attribute condition (see Ex. 6.3).

A policy can be arbitrary restrictive in general. In the context of model synchronization, we need to ensure that the propagation operations are still defined for all valid inputs. For this reason, we introduce the notion of a *conservative* policy. In the case of forward propagation, a policy for the set of forward translation rules is conservative, if all valid source models can be translated.

Definition 6.1 (Policy for Operational Translation Rules) *Given a TGG and let* TR_{FT} *be the derived set of forward translation rules. A* policy $pol : TR_{FT} \rightarrow TR'_{FT}$ *for restricting the applications of the rules in* TR_{FT} *maps each rule* $tr_{FT} \in TR_{FT}$ *to an extended rule* $tr'_{FT} \in TR'_{FT}$*, where* tr'_{FT}

is given by tr_{FT} extended by a set of additional attribute conditions $AttC_{pol}(tr_{FT})$. The policy pol is called conservative*, if the derived model transformation relation $MTR'_{FT} \subseteq VL_S \times VL_T$ based on TR'_{FT} is left total and is contained in the model transformation relation MTR_{FT} derived from TR_{FT}, i.e., $MTR'_{FT} \subseteq MTR_{FT}$.*

A policy for backward translation rules TR_{BT} is defined analogously by replacing FT with BT and it is conservative if the derived model transformation relation $MTR'_{BT} \subseteq VL_T \times VL_S$ is left total and contained in MTR_{BT}.

In order to automatically check that a policy is conservative we provide a sufficient condition by Lem. 6.2 below based on the analysis of dependencies between rules [11]. Intuitively, two transformation steps $G_0 \overset{p_1,m_1}{\Longrightarrow} G_1 \overset{p_2,m_2}{\Longrightarrow} G_2$ are sequentially independent, if (1) there is no use-delete dependency (the first step uses (creates or reads) an element (node, edge, or attribute) that is deleted by p_2 in the second step) and (2) there is no forbid-produce dependency. A produce-forbid dependency occurs if the first step forbids a pattern by a negative application condition of p_1 and the second step produces some elements of it, such that applying the second step first will disable the execution of the first step thereafter.

A policy restricts the applicability of rules. The main challenge is to ensure that the restrictions are not too strict. In more detail, for each valid input model of an operational transformation sequence we have to ensure that there is an equivalent transformation sequence respecting all restrictions of the policy. The key idea is to check for each restriction of a rule p whether there are rules that could depend on the execution of p. If we can show that there is no dependency to all possible subsequent steps in an operational transformation sequence, we can conclude that all steps via p can be shifted to the end of the sequence. This allows us to focus on p itself. As stated by Lem. 6.2 below, it is then sufficient to show that for each match of p there is an equivalent match satisfying the conservative policy.

Lemma 6.2 (Conservative Policy) *Let $pol : TR_{FT} \to TR'_{FT}$ be a policy, such that for each rule $tr'_{FT} = pol(tr_{FT})$ in TR'_{FT} with $tr : L \to R$ the following condition holds.*

1. *Given a match $m : L \to G$ for tr_{FT}, then there is also a match $m' : L \to G$ for tr'_{FT} satisfying $AttC_{pol}(tr_{FT})$.*
2. *If $AttC_{pol}(tr_{FT}) \neq \emptyset$, then for each rule $tr_2 \in TR_{FT}$ with $tr_{FT} \neq tr_2$ the pair (tr_{FT}, tr_2) is sequentially independent.*

Then, the policy pol is conservative (cf. Def. 6.1). A similar fact holds for a policy $pol : TR_{BT} \to TR'_{BT}$ concerning backward translation rules.

Proof (Idea) According to Def. 6.1, policy *pol* is conservative, if the derive model transformation relation MTR'_{FT} is left total. The model transformation relation MTR based on TR_{FT} is left total due to the completeness result for TGG model transformations based on forward translation rules (cf. Thm. 1 in [22]). Thus, given a source model $G^S \in VL_S$, there is a complete forward translation sequence s_{FT} via TR_{FT}. We have to show that there is also a complete forward translation sequence s'_{FT} via TR'_{FT}. First of all, $MTR'_{FT} \subseteq MTR_{FT}$, because the additional attribute conditions only restrict the possible transformation sequences and no additional ones are possible. Item (1) in Lem. 6.2 ensures that for each step $s_{i,FT}$ in s_{FT} via TR_{FT}, there is a step $s'_{i,FT}$ via TR'_{FT}, but this step may differ on the resulting triple graph. However, item (2) ensures that there is no subsequent step in s_{FT} via a different rule that is sequentially dependent to neither $s_{i,FT}$ nor $s'_{i,FT}$. Therefore, we can iteratively exchange the original steps with corresponding ones via TR'_{FT}, shift them to the end of the the sequence, and continue with the next step that is not via TR'_{FT}. Finally, we derive a complete forward translation sequence s'_{FT} via TR'_{FT}. For the full proof see Fact 7 in [24]. □

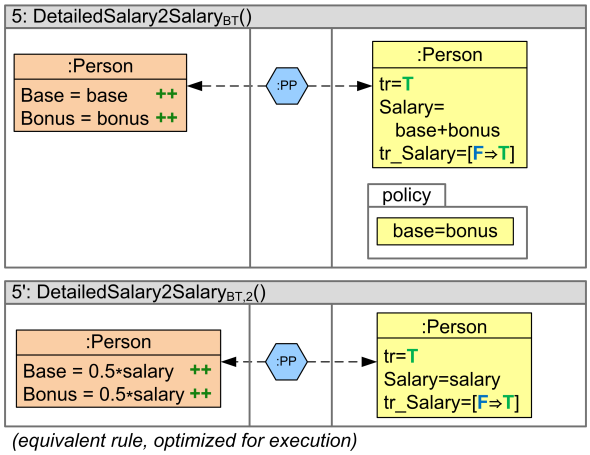

Fig. 6.1 Backward Translation Rule without (5) and with (5′) conservative policy

Example 6.3 (Conservative Policy) In Fig. 6.1, the backward translation rule 5 : "*DetailedSalary2Salary$_{BT}$*()" from Ex. 5.6 is extended to the rule 5′ : "*DetailedSalary2Salary$_{BT,2}$*()" by a policy in the form of an additional positive application condition in order to ensure determinism. Since the left hand side of this rule specifies only the sum of the salary of a person, the values of the base and bonus components are not fixed via a match. The positive application condition (PAC) [11] requires that both values are set to half times the amount of the salary sum. Now, this is possible for each number, such that we can conclude that the policy is conservative (Lem. 6.2), which is important for ensuring completeness of the propagation operation bPpg (see Thm. 8.2).

In order to ensure termination of the propagation operations, we restrict the sets of operational rules to those that modify at least one translation attribute. We call these rules kernel translation rules. In the case of forward translation rules, the kernel forward translation rules $TR^{+s}_{FT} \subseteq TR_{FT}$ are those forward translation rules that are derived from the source creating triple rules $TR^{+s} \subseteq TR$ of the triple rules TR. The remaining forward translation rules $TR^{1s}_{FT} = TR_{FT} \setminus TR^{+s}_{FT}$

are those derived from the source identic triple rules TR^{1s}. Vice versa, the kernel backward translation rules $TR^{+t}_{BT} \subseteq TR_{BT}$ are the backward translation rules that are derived from the target creating triple rules $TR^{+t} \subseteq TR$ and TR^{1t}_{BT} are the remaining backward translation rules derived from the target identic triple rules. Finally, the kernel consistency creating triple rules, however, are given by the complete set of consistency creating rules TR_{CC}.

The restriction of the set of operational rules can possibly cause that for a valid input model, there is no longer a valid operational transformation sequence via forward or backward translation rules, respectively. However, we can use the same idea as before and check that the remaining rules do not depend on the omitted ones (TR^{1s}_{FT} and TR^{1t}_{BT}) as stated by Rem. 6.4 below. This ensures that the rules that do not change any translation attribute can be shifted to the end of each sequence and thus, can be omitted while still all valid input models can be processed successfully.

Remark 6.4 (Shifting of Independent Steps) Given two sets P_1 and P_2 of rules such that each pair $(p_1, p_2) \in P_1 \times P_2$ is sequentially independent. Then, there is a transformation sequence ($G \overset{r^*}{\Longrightarrow} H$) via ($P_1 \cup P_2$) if and only if there are transformation sequences: $s_1 = (G \overset{p^*}{\Longrightarrow} G_1)$ via P_2 and $s_2 = (G_1 \overset{q^*}{\Longrightarrow} H)$ via P_1 with same G_1. This result is shown by Fact 3 in App. A.2 in [24].

Now, we come to the most important property that has to be checked for the operational translation rules in order to ensure correct propagation operations—*deterministic behavior*. First of all, this means that their execution has *functional behavior*, i.e., ensures unique results. In addition to that, their execution does not require *backtracking*. This means that once an operational translation rule is applied, we do not have to undo the step during the synchronization process. Moreover, we have to ensure termination. For this purpose, we introduce the notion of kernel-grounded operational translation rules and show thereafter that this property allows us to restrict the sets of rules appropriately.

Definition 6.5 (Kernel-Grounded and Deterministic Sets of Operational Translation Rules) *Let $TGG = (TG, \emptyset, TR)$ be a triple graph grammar, from which we obtain the operational translation rules TR_{CC}, TR_{FT}, and TR_{BT}. They are called kernel-grounded, if the pairs $(TR^{1s}_{FT}, TR^{+s}_{FT})$ and $(TR^{1t}_{BT}, TR^{+t}_{BT})$ are sequentially independent. This means that there is no pair (p_1, p_2) of sequentially dependent rules with either $(p_1, p_2) \in (TR^{1s}_{FT} \times TR^{+s}_{FT})$ or $(p_1, p_2) \in (TR^{1t}_{BT} \times TR^{+t}_{BT})$.*

The sets of operational translation rules TR_{CC}, TR_{FT}, and TR_{BT} (possibly extended by conservative policies) are called deterministic*, if they have functional behavior and do not require backtracking.*

The tool AGG [37] supports the automated analysis of dependencies between rules. We apply this analysis engine to check whether a policy is conservative and that the reduced sets of operational rules are sufficient to ensure completeness of the propagation operations.

In order to check that the sets of operational translation rules are kernel-grounded and deterministic, we first describe how the preconditions of Def. 6.5 are checked using the tool AGG.

1. Sequential independence of the pairs $(TR^{1s}_{FT}, TR^{+s}_{FT})$ and $(TR^{1t}_{BT}, TR^{+t}_{BT})$: we can use the tool AGG for the analysis of rule dependencies based on the generation of critical pairs according to Fact 2 in [24].
2. Applied policies are conservative: According to Lem. 6.2. This requires that the additional application conditions according to the policy restricts the evaluation of attribute values only, i.e., the assignment of variables. We have to show that the existence of matches is preserved for each rule and that other rules are not sequentially dependent. For the latter, we can again use the tool AGG and validate that the corresponding table entries show the value 0. The preservation of the existence of matches can be ensured by checking that the affected variables are free in the unmodified rule (tr_{FT} or tr_{BT}), i.e., they are not part of a term that is connected to a node in the LHS (L_{FT} or L_{BT}).

Moreover, we can apply the presented results for showing that the derived model transformation relations are left total. This is the basis to ensure that the propagation operations are left total.

Remark 6.6 (Left Totality) If the sets of operational translation rules of a TGG are kernel-grounded, we can conclude that the forward model transformation relations $MTR_F: VL(TG^S) \Rightarrow VL(TG^T)$ based on TR^{+s}_{FT} and the backward model transformation $MT_B: VL(TG^T) \Rightarrow VL(TG^S)$ based on TR^{+s}_{FT} specify left total relations as shown by Fact 5 in [24]. This means that the model transformations can be performed on reduced sets of operational translation rules. Source identic triple rules TR^{1s}_{FT} are not used for forward translations and target identic triple rules TR^{1t}_{BT} are not used for backward translations. According to Def. 6.1, we can specify conservative policies in order to reduce the amount of possible transformation sequences and derive left total model transformation relations MTR'_{FT} and MTR'_{BT} that use these policies.

In order to check that the derived sets of operational translation rules have functional behavior and do not require backtracking, we first show by Lem. 6.7 below that we can ensure termination if all operational translation rules modify at least one translation attribute. We generally assume that the input models are finite on the structure part, i.e., the carrier sets of the data values can be infinite, but the graph nodes and all sets of edges are finite.

Lemma 6.7 (Termination) *Let $TGG = (TG, \emptyset, TR)$ be a triple graph grammar, where TR does not contain a trivial triple rule $tr = (L \to R)$ with $L = R$. Let further TR_{CC}, TR^{+s}_{FT}, and TR^{+t}_{BT} be the derived sets of operational translation rules for consistency creating, forward translation and backward translation, respectively according to Def. 5.1 and possibly extended by some policies. Then, the transformation systems*

TR_{CC}, TR^{+s}_{FT}, and TR^{+t}_{BT} are terminating for any finite input triple graph.

Proof (Idea) As described in Rem. 5.10, an operational translation rule may change the value of a translation attribute from **F** to **T**, but not vice versa. Since the amount of elements that are marked with translation attributes is not changed by any operational translation rule and the input models are finite, we can conclude that any transformation sequence via the given sets terminates. For the full proof see Fact 8 in [24], which is based on [22]. □

Functional behavior of a transformation system means that the execution of the system yields unique results. In the context of model transformations, the execution may involve backtracking. Consider, e.g., a forward translation sequence where no further operational rule is applicable, but the source model is not completely translated. In this case, further sequences may exist, because of non-determinism concerning the choice of possible transformation rules and matches.

A system of operational translation rules has functional behavior and does not require backtracking, if all significant critical pairs are strictly confluent as shown by Fact 9 in [24] based on a corresponding result for forward translation rules in [22].

Remark 6.8 (Analysis of Functional Behavior and Backtracking) The tool AGG [37] provides an analysis engine for generating the complete set of critical pairs as described in [22] and we provide the analysis results for our example TGG in Sec. 7. A critical pair $(P_1 \xleftarrow{p_1,m_1} K \xrightarrow{p_2,m_2} P_2)$ consists of two parallel dependent transformation steps. It is significant, if it can be embedded in a transformation sequence via operational translation rules starting at a valid input model (cf. Def. 17 in [24]). A critical pair concerning a forward model transformation is not significant, if the source component of K cannot be embedded in a valid source model, because changes to the source component only occur on translation attributes. The dual result holds for backward model transformations.

Strict confluence of a critical pair requires that we provide sequences of transformation steps $(P_1 \overset{*}{\Rightarrow} H \overset{*}{\Leftarrow} P_2)$ solving the conflict $(P_1 \xleftarrow{p_1,m_1} K \xrightarrow{p_2,m_2} P_2)$ in a compatible and NAC-consistent way [11]. This means that any element that is preserved in $(P_1 \xleftarrow{p_1,m_1} K \xrightarrow{p_2,m_2} P_2)$ is also preserved in $(P_1 \overset{*}{\Rightarrow} H \overset{*}{\Leftarrow} P_2)$. If no critical pair exists at all, we directly derive that the system has functional behavior and does not require backtracking.

7 Synchronization Based on TGGs

This section shows how to construct the operation fPpg of a TGG synchronization framework (cf. Def. 3.4) as a composition of auxiliary operations ⟨fAln, Del, fAdd⟩. Symmetrically, operations ⟨bAln, Del, bAdd⟩ are used to define the operation bPpg. As a general requirement, the given TGG has to provide *deterministic* sets of operational translation rules, meaning that the algorithmic execution of the forward translation, backward translation, and consistency creating rules ensures functional behavior (unique results) and does not require backtracking. For this purpose, additional policies can be defined that restrict the matches of operational translation rules as presented in Sec. 6 by Lem. 6.2. Rem. 6.8 in Sec. 6 provides sufficient conditions for deterministic operational translation rules. We provide additional static conditions and automated checks in the technical report [24].

The general synchronization process is performed as follows (see Def. 7.1 and Fig. 7.1, where we use double arrows (↔) for correspondence in the signature of the operations, and the explicit triple graphs for the construction details). Given two corresponding models G^S and G^T and an update of G^S via the graph modification $a = (G^S \xleftarrow{a_1} D^S \xrightarrow{a_2} G'^S)$ with $G'^S \in VL_S$, the forward propagation fPpg of model update a is performed in three steps via the auxiliary operations fAln, Del, and fAdd. At first, the deletion performed in a is reflected into the correspondence relation between G^S and G^T by calculating the forward alignment remainder via operation fAln. This step deletes all correspondence elements whose elements in G^S have been deleted. In the second step, performed via operation Del, the two maximal subgraphs $G^S_k \subseteq G^S$ and $G^T_k \subseteq G^T$ are computed such that they form a consistent integrated model in $VL(TGG)$ according to the TGG. All elements that are in G^T but not in G^T_k are deleted, i.e., the new target model is given by G^T_k. Finally, in the last step (operation fAdd), the elements in G'^S that extend G^S_k are transformed to corresponding structures in G'^T, i.e., G^T_k is extended by these new structures. The result of fAdd, and hence also fPpg, is an integrated model $G' = (G'^S \leftrightarrow G'^T)$. Since graph transformation is non-deterministic in general, we require that the sets of operational translation rules are deterministic in order to ensure unique results for both, the second and the third step of propagation operation fPpg.

Definition 7.1 (Auxiliary TGG Operations) *Let $TGG = (TG, \emptyset, TR)$ be a TGG with deterministic sets TR_{CC}, TR^{+s}_{FT}, and TR^{+s}_{BT} of operational translation rules and let further $MF(TGG)$ be the derived TGG model framework.*

1. *The auxiliary operation* fAln *computing the forward alignment remainder is given by* $\mathsf{fAln}(r, a) = r'$*, as specified in the upper part of Fig. 7.1. The square marked by (PB) is a pullback [11], meaning that D^C is the intersection of D^S and G^C.*
2. *Let $r = (s, t): G^S \leftrightarrow G^T$ be a correspondence relation, then the result of the auxiliary operation* Del *is the maximal consistent subgraph $G^S_k \leftrightarrow G^T_k$ of r, given by* $\mathsf{Del}(r) = (a, r', b)$*, which is specified in the middle part of Fig. 7.1.*
3. *Let $r = (s, t): G^S \leftrightarrow G^T$ be a consistent correspondence relation, $a = (1, a_2) : G^S \to G'^S$ be a source modification and $G'^S \in VL_S$. The result of the auxiliary operation* fAdd*, for propagating the additions of source modification a, is a consistent model $G'^S \leftrightarrow G'^T$ extending*

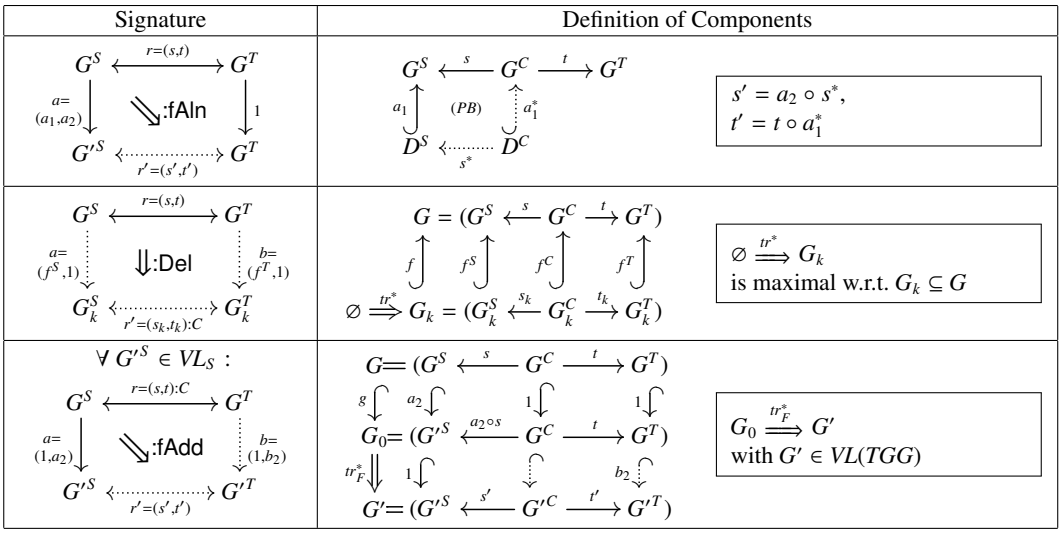

Fig. 7.1 Auxiliary operations fAln, Del and fAdd

$G^S \leftrightarrow G^T$, and is given by fAdd$(r, a) = (r', b)$*, according to the lower part of Fig. 7.1.*

Remark 7.2 (Auxiliary TGG Operations) Intuitively, operation fAln constructs the new correspondence graph D^C from the given G^C by deleting all correspondence elements in G^C whose associated elements in G^S are deleted via update a and, for this reason, do not occur in D^S. Operation Del is executed by applying consistency creating rules (cf. Sec. 5) to the given integrated model until no rule is applicable any more. If, at the end, $G^S \leftrightarrow G^T$ is completely marked, the integrated model is already consistent; otherwise, the result is the largest consistent integrated model included in $G^S \leftrightarrow G^T$. Technically, the application of the consistency creating rules corresponds to a maximal triple rule sequence as shown in the right middle part of Fig. 7.1 and discussed in more detail in [23]. Finally, fAdd is executed by applying forward translation rules (cf. Sec. 4 and 5) to $G'^S \leftrightarrow G^T$ until all the elements in G'^S are marked. Intuitively, these TGT steps form a model transformation of G'^S extending G^T. Technically, the application of the forward translation rules corresponds to a source-consistent forward sequence from G_0 to G', as shown in the right lower part of Fig. 7.1. By correctness of model transformations [12], the sequence implies consistency of G' as stated above. The constructions for these auxiliary operations are provided in full detail in [24]. Note that the constructions for Del and fAdd yield unique results due to the requirement that the operational translation rules are deterministic (cf. Def. 7.1).

Auxiliary operation Del is based on the execution of consistency creating rules. The computed resulting triple graph G_k is required to be consistent ($G_k \in VL$). This result is ensured by the equivalence of maximal triple and complete extended consistency creating sequences according to Rem. 7.3 below and shown by Fact 11 in [24].

Remark 7.3 (Equivalence of Maximal Triple and Complete Extended Consistency Creating Sequences) Given a set of non-identic consistency creating rules TR_{CC} and $G \in VL(TG)$. Then, the following are equivalent for almost injective matches.

1. There is a TGT-sequence $s = (\varnothing \overset{tr^*}{\Longrightarrow} G_k)$ via TR with injective embedding $f : G_k \to G$, such that s is f-maximal, i.e., any extension of s via TR is not compatible with f.
2. There is a terminated consistency creating sequence $s' = (G'_0 \overset{tr^*_{CC}}{\Longrightarrow} G'_k)$ via TR_{CC} with $G'_0 = Att^{\mathbf{F}}(G)$, i.e., all translation attributes are set to **F**.

Moreover, the sequences correspond via $G'_k = H \oplus Att^{\mathbf{T}}_{G_k} \oplus Att^{\mathbf{F}}_{H \setminus G_k}$.

Example 7.4 (Forward Propagation via Operation fPpg*)* Figure 7.2 shows the application of the three steps of synchronization operation fPpg to the visual models of our running example. After removing the dangling correspondence node of the alignment in the first step (fAln), the maximal consistent subgraph of the integrated model is computed (Del) by stepwise marking the consistent parts: consistent parts are indicated by gray boxes with checkmarks in the visual notation and by bold font faces in the graph representation. Note that node "Bill Gates" is part of the target graph in this maximal consistent subgraph, even though it is not in correspondence with any element of the source graph. This is possible, because node "Bill Gates" is now connected to a different department (cf. rule 8:Empty2OtherP in Fig. 3.4). Moreover, attributes Base and Bonus of Melinda Gates in the source component are not marked, because they are inconsistent with attribute Salary according to triple rule 6:Detailed-Salary2Salary in Sec. 5 (base + bonus ≠ Salary). In the final step (fAdd), the inconsistent elements in the target model are removed and the remaining new elements of the update are propagated towards the target model by model transformation, such that all elements are finally marked as consistent.

The constructions for the auxiliary operations fAln, Del, and fAdd provide the basis for the propagation operation

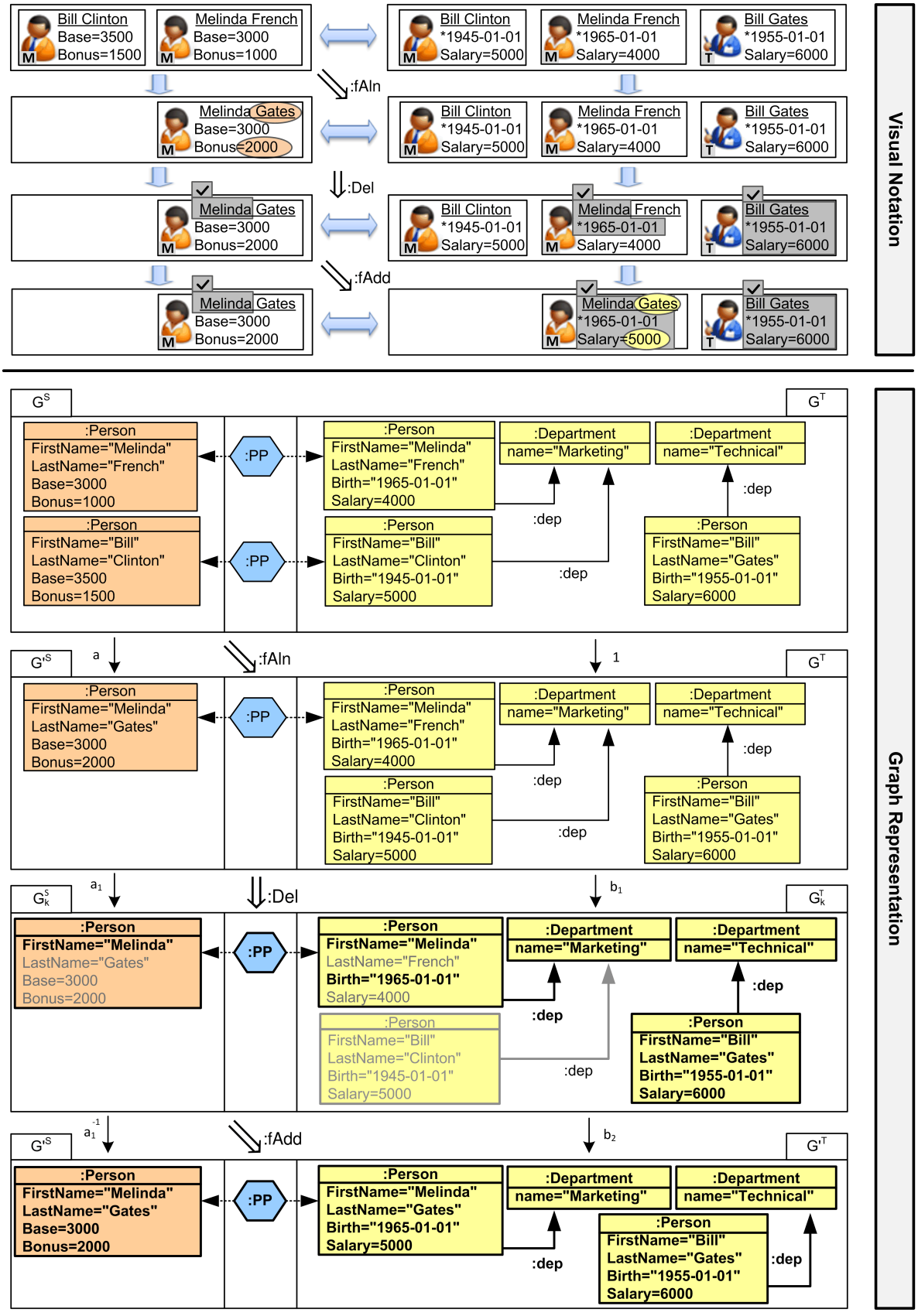

Fig. 7.2 Forward propagation in detail: visual notation (top) and graph representation (bottom)

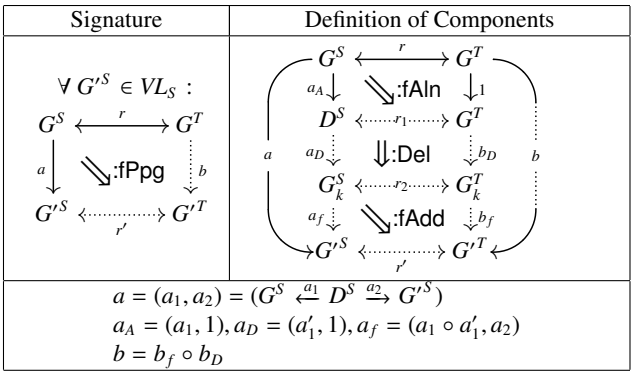

Fig. 7.3 Synchronization operation fPpg - formal definition

fPpg. Together with its symmetric version, namely the backward propagation operation bPpg, we derive the TGG synchronization framework according to Def. 7.5. Forward and backward propagation operations fPpg and bPpg are called *complete*, if they yield valid results for any valid input. Completeness of the synchronization operations is an important property in the context of TGGs and therefore, it is worth to emphasize it explicitly, while it is implicitly included already within the signature in Fig. 7.3.

Definition 7.5 (Derived TGG Synchronization Framework) *Let $TGG = (TG, \emptyset, TR)$ be a TGG with deterministic sets TR_{CC}, TR^{+s}_{FT}, and TR^{+s}_{BT} of derived operational translation rules and with derived model framework $MF(TGG)$, then operation* fPpg *of the* derived TGG synchronization framework *Synch(TGG) is given by the composition of auxiliary operations (*fAln, Del, fAdd*) as described in Rem. 7.6 according to Fig. 7.3. Symmetrically—not shown explicitly—we obtain* bPpg *as composition of auxiliary operations (*bAln, Del, bAdd*). Synch(TGG) is called* complete*, if its propagation operations are complete, i.e., they always yield a result for any valid input.*

```
1  /* == alignment remainder == */
2  forall(correspondence nodes without image
3         in the source model){
4     delete these elements }
5  /* ==== delete === */
6  while(there is a triple rule p such that
7        R\L is unmarked){
8     apply to G the consistency creating
9     rule corresponding to p }
10 forall(unmarked nodes and edges from the
11        target model){
12    delete these elements }
13 /* ===== add ===== */
14 while(there is a forward translation
15       rule applicable to G){
16    apply to G the forward translation
17    rule }
```

Fig. 7.4 Synchronization operation fPpg - algorithm

Remark 7.6 (Construction of fPpg *according to Fig. 7.3)* Given a not necessarily consistent integrated model $r: G^S \leftrightarrow G^T$ and source model update $a: G^S \to G'^S$ with $G'^S \in VL_S$, we compute $\mathsf{fPpg}(r, a)$ as follows. First, fAln computes the correspondence $(D^S \leftrightarrow G^T)$, where D^S is the part of G^S that is preserved by update a. Then, Del computes its maximal consistent integrated submodel $(G^S_k \leftrightarrow G^T_k)$. Finally, fAdd composes the embedding $G^S_k \to G'^S$ with correspondence $(G^S_k \leftrightarrow G^T_k)$ leading to $(G'^S \leftrightarrow G^T_k)$, which is then extended into the integrated model $(G'^S \leftrightarrow G'^T)$ via forward transformation. If $G'^S \notin VL_S$, then the result is given by $b = (1,1): G^T \to G^T$ together with the correspondence relation $r' = (\emptyset, \emptyset)$ and additionally, an error message is provided. Fig. 7.4 describes this construction algorithmically in pseudo code, leaving out the error handling; marking is explained in Sec. 5.

Fact 7.7 (Case Study: Termination of Synchronization Operations) *The derived synchronization operations* fPpg *and* bPpg *for our example TGG terminate.*

Proof The TGG does not contain any trivial rule $tr: L \to L$. According to Def. 7.5, the synchronization operations are based on the sets TR_{CC}, TR^{+s}_{FT}, and TR^{+s}_{BT} of operational translation rules. Hence, we can apply Lem. 6.7 and derive that the synchronization operations are terminating. □

By Fact 7.7, we know that the synchronization operations are terminating. This allows us to use AGG to generate the critical pairs (see Fig. 7.5) in order to check that the operations are deterministic and that the derived synchronization framework is correct and complete using Thm. 8.1 and Thm. 8.2.

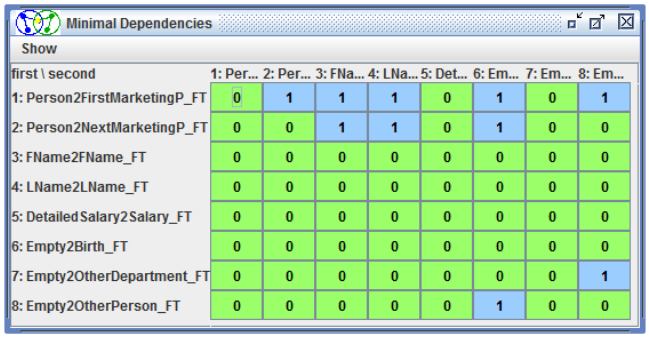

Fig. 7.5 Dependency analysis with AGG for TR_{FT} - fields with "1" contain dependencies

Fact 7.8 (Case Study: Determinism) *The derived sets of operational rules for* fPpg *and* bPpg *of our example TGG are deterministic and kernel-grounded.*

Proof In order to show that the synchronization operations are deterministic, we can apply Thm. 8.1 by showing that the sets of operational translation rules are kernel-grounded, terminating, and all significant critical pairs are strictly confluent using that they are terminating according to Fact 7.7.

Minimal Conflicts

Show

first \ second	1: Per...	2: Per...	3: FNa...	4: LNa...	5: Deta...	6: Emp...	7: Emp...	8: Emp...
1: Person2FirstMarketingP_CC	1	0	0	0	0	0	0	0
2: Person2NextMarketingP_CC	0	0	0	0	0	0	0	0
3: FName2FName_CC	0	0	0	0	0	0	0	0
4: LName2LName_CC	0	0	0	0	0	0	0	0
5: DetailedSalary2Salary_CC	0	0	0	0	0	0	0	0
6: Empty2Birth_CC	0	0	0	0	0	0	0	0
7: Empty2OtherDepartment_CC	0	0	0	0	0	0	0	0
8: Empty2OtherPerson_CC	0	0	0	0	0	0	0	0

Fig. 7.6 Critical pair analysis with AGG for TR_{CC} - fields with with "1" contain conflicts

Minimal Conflicts

Show

first \ second	1: Per...	2: Per...	3: FNa...	4: LNa...	5: Det...	6: Em...	7: Em...	8: Em...
1: Person2FirstMarketingP_FT	1	0	0	0	0	0	0	0
2: Person2NextMarketingP_FT	0	0	0	0	0	0	0	0
3: FName2FName_FT	0	0	0	0	0	0	0	0
4: LName2LName_FT	0	0	0	0	0	0	0	0
5: DetailedSalary2Salary_FT	0	0	0	0	0	0	0	0
6: Empty2Birth_FT	0	0	0	0	0	0	0	0
7: Empty2OtherDepartment_FT	0	0	0	0	0	0	0	0
8: Empty2OtherPerson_FT	0	0	0	0	0	0	0	0

Fig. 7.7 Critical pair analysis with AGG for TR_{FT} - fields with "1" contain conflicts

Minimal Dependencies

Show

first \ second	1: Per...	2: Per...	3: FNa...	4: LNa...	5: Det...	6: Em...	7: Em...	8: Em...
1: Person2FirstMarketingP_BT	0	1	1	1	1	1	0	0
2: Person2NextMarketingP_BT	0	0	1	1	1	1	0	0
3: FName2FName_BT	0	0	0	0	0	0	0	0
4: LName2LName_BT	0	0	0	0	0	0	0	0
5: DetailedSalary2Salary_BT	0	0	0	0	0	0	0	0
6: Empty2Birth_BT	0	0	0	0	0	0	0	0
7: Empty2OtherDepartment_BT	0	0	0	0	0	0	0	1
8: Empty2OtherPerson_BT	0	0	0	0	0	0	0	0

Fig. 7.8 Dependency analysis with AGG for TR_{BT} - fields with "1" contain dependencies

Minimal Conflicts

Show

first \ second	1: Per...	2: Per...	3: FNa...	4: LNa...	5: Det...	6: Emp...	7: Emp...	8: Emp...
1: Person2FirstMarketingP_BT	1	0	0	0	0	0	0	0
2: Person2NextMarketingP_BT	0	0	0	0	0	0	0	0
3: FName2FName_BT	0	0	0	0	0	0	0	0
4: LName2LName_BT	0	0	0	0	0	0	0	0
5: DetailedSalary2Salary_BT	0	0	0	0	0	0	0	0
6: Empty2Birth_BT	0	0	0	0	0	0	0	0
7: Empty2OtherDepartment_BT	0	0	0	0	0	0	0	0
8: Empty2OtherPerson_BT	0	0	0	0	0	0	0	0

Fig. 7.9 Critical pair analysis with AGG for TR_{BT} - fields with "1" contain conflicts

Concerning the set TR_{FT}, we used AGG to derive the dependency table depicted in Fig. 7.5. The source identic rules are the rules with number 6 to 8. There is no dependency (entry > 0) for any pair (p, q) with $p \geq 6$ and $q \leq 5$. Moreover, there are no target identic backward translation rules, because all triple rules are creating on the target component. Therefore, the sets of operational translation rules are kernel-grounded (see Def. 6.5).

By Fact 7.7, we know that the transformation systems based on the operational translation rules are terminating. We analyzed the critical pairs using the critical pair analysis engine of AGG. Concerning the set TR_{CC}, we derived the resulting table depicted in Fig. 7.6. The only generated critical pair is (p_1, p_1) for $p_1 = Person2FirstMarketingP_{CC}$ and it is strictly confluent by applying rule $p_2 = Person2NextMarketingP_{CC}$ to the remaining structure and since p_2 does not contain any NAC we automatically have strict confluence.

Concerning the set TR_{FT}, we derived the resulting table depicted in Fig. 7.7, where we used the constraint that there are no two departments with name "Marketing". This is always ensured for the language $VL(TGG)$ due to the NACs of the first two rules. The only significant critical pair is strictly confluent via one transformation step using rule $p_2 = Person2NextMarketingP_{FT}$, where no NAC is involved.

The set TR_{BT} is not functional, because there is a choice of how to split the salary into base and bonus. We can restrict the choice for the rule "DetailedSalary2Salary" to $base = bonus = 1/2 \cdot salary$ as a policy, which is shown by the additional positive application condition in Fig. 5.6. We can apply Lem. 6.2 and derive that the policy is conservative. First of all, no other rule depends on this rule, which we verified by the generated dependency table by AGG in Fig. 7.8. Moreover, any match for the original rule implies that there is a match for the restricted rule, because the restricted values are real numbers and, therefore, always possible. We derive the table of generated critical pairs depicted in Fig. 7.9, where the only significant critical pair is again strictly confluent via one transformation step using rule $p_2 = Person2NextMarketingP_{BT}$, where no NAC is involved.

Summing up, the sets of operational translation rules are kernel-grounded and all significant critical pairs are strictly confluent, such that we can apply Thm. 8.1 and derive that the derived sets of operational rules are deterministic. □

8 Correctness and Invertibility

In this section, we present our main results that show the correctness, completeness and invertibility of our synchronization framework. According to Def. 3.4, correctness requires that the synchronization operations are deterministic, i.e., they have functional behavior (cf. Sec. 5) and ensure laws $(a1)$ - $(b2)$. Concerning the first requirement, i.e., that the synchronization operations are deterministic, Thm. 8.1 below provides a sufficient condition based on the notion of critical pairs [11]. In order to ensure this condition, Sec. 6 presents

the concept of additional propagation policies that eliminate non-determinism. They can be seen as a kind of application conditions for the rules and are called *conservative*, if they preserve the completeness result. Lem. 6.2 provides a sufficient static condition for checking this property and we performed the automated analysis of this condition for our example TGG using the tool AGG [37] as described in Sec. 6. Note again that we generally require almost injective matching (cf. Sec. 3).

Theorem 8.1 (Deterministic Synchronization Operations) *Let TGG be a triple graph grammar that does not contain an identical rule $tr : L \to L$. If the significant critical pairs of the sets of operational translation rules are strictly confluent and the systems of rules are terminating, then the sets of operational translation rules are deterministic (see Def. 6.5), which implies that the derived synchronization operations* fPpg *and* bPpg *are deterministic as well.*

Proof (Idea) Operations fAln and bAln are given by pullback construction, which is unique up to isomorphism by definition. Therefore, they are deterministic. Termination of Del, fAdd, and bAdd is ensured according to Rem. 5.10, because *TGG* does not contain an identical triple rule and the operational translation rules are given by TR_{CC}, TR^{+s}_{FT}, and TR^{+s}_{BT}. By Rem. 6.8 we know that functional behavior of the transformation systems is ensured and backtracking is not required, if all significant critical pairs are strictly confluent and the system is terminating. This ensures that operations Del, fAdd, and bAdd are deterministic. Thus, operations fPpg and bPpg are deterministic. For the full proof see Fact 1 in [24]. □

A correct synchronization framework has to satisfy laws $(a1) - (b2)$ in Def. 3.4. Intuitively, the propagation operations have to preserve consistent inputs. First of all, if the given integrated model is already consistent and the given update does not change anything, then the resulting integrated model has to be the given one and the resulting update on the opposite domain has to be the identity (laws $(a1)$ and $(b1)$). Most importantly, given an arbitrary integrated model together with a source update $d^S : G^S \to G'^S$ with consistent new source model $G'^S \in VL^S$, then the forward propagation via fPpg has to provide a new consistent integrated model $G'^S \leftrightarrow G'^T \in VL$. *Completeness* of a synchronization framework *Synch*(*TGG*) requires that operations fPpg and bPpg can be successfully applied to all consistent source models $G'^S \in VL_S$ and target models $G'^T \in VL_T$, respectively. This property is of general importance in the context of TGGs and therefore, we explicitly show it together with correctness in Thm. 8.2 below. Both results are ensured, if the sets of operational rules are deterministic as in Thm. 8.1 and additionally, if they are kernel-grounded (cf. Def. 6.5), i.e., the effective forward and backward translation rules do not depend on any source or target identic translation rule, respectively. This second condition is important for laws $(a1) - (b2)$, because it ensures that the computed transformation sequences via auxiliary operations Del, fAdd, and bAdd can be composed in a consistent way.

Theorem 8.2 (Correctness and Completeness) *Let Synch*(*TGG*) *be a derived TGG synchronization framework, such that the sets of operational translation rules of TGG are kernel-grounded and deterministic (see Def. 6.5). Then Synch*(*TGG*) *is correct and complete.*

Proof (Idea) By Thm. 8.1, the provided constructions of operations fPpg and bPpg based on the operational translation rules have functional behavior, i.e., for each input the computation yields a unique output. Thus, the derived synchronization framework is complete.

In order to show correctness, we have to show laws $(a1)$ and $(a2)$ of Def. 3.4. Precondition $G \in VL$ of law $(a1)$ implies that there is a triple sequence $\varnothing \overset{tr^*}{\Longrightarrow} G$ via TR and by Rem. 7.3, there is a corresponding complete consistency creating sequence. Moreover, there is a corresponding forward translation sequence via TR_{FT} by Thm. 1 in [22]. Using the precondition that the operational translation rules are kernel-grounded, we can conclude that all steps via TR^{1s}_{FT} can be shifted to the end. Thus, no further forward translation rule in TR^{+s}_{FT} is applicable. The functional behavior of operation fPpg and the given identical source update $d_s = id_{G^S}$ ensure the requested result, i.e., we derive target update $d^T = id_{G^T}$ and the integrated model $G' = G$. In order to show law $(a2)$, we can use precondition $G'^S \in VL_S$, which implies that there is a source consistent forward sequence s_F starting at G'^S and a corresponding complete forward translation sequence. Since the operational rules are kernel-grounded we can conclude by Rem. 6.6 that there is a complete forward translation sequence s^{+s}_{FT} via TR^{+s}_{FT}. Due to functional behavior of operation Del we derive a consistency creating sequence that corresponds to the first part of s_F and therefore, to a sequence s_{FT} via forward translation rules. Since the sets of operational rules are kernel-grounded, we can conclude that the steps via TR^{+s}_{FT} do not depend on TR^{1s}_{FT}. This allows us to complete s_{FT} using TR^{+s}_{FT}, where we can shift the source identic steps via TR^{1s}_{FT} to the end. Thus, we derive a complete forward translation sequence, where we can omit the steps via TR^{1s}_{FT} at the end. Functional behavior of TR^{+s}_{FT} implies that this sequence corresponds to the complete forward translation sequence s^{+s}_{FT} and therefore, to a source consistent forward sequence s^{+s}_F leading to G'. Thus, $G' \in VL$ by Thm. 2 in [17]. For the full proof see Lemma 3 in [24]. □

Example 8.3 (Correctness and Completeness) The initially derived set of backward transformation rules for our running example is not completely deterministic because of the non-deterministic choice of base and bonus values for propagating the change of a salary value. Therefore, we defined a conservative policy for the responsible backward triple rule by fixing the propagated values of modified salary values to $bonus = base = 0.5 \times salary$. By Lem. 6.2 in Sec. 6, we provided a sufficient static condition for checking that a policy is conservative; we validated our example and showed that the derived sets of operational rules for fPpg and bPpg are deterministic and kernel-grounded (cf. Fact 7.8 in Sec. 7). For this reason, we can apply Thm. 8.2 and conclude that the derived

TGG synchronization framework is correct and complete (cf. Fact 8.4 below).

Fact 8.4 (Case Study: Correctness and Completeness) *The derived synchronization framework for our example TGG is correct and complete.*

Proof By Fact 7.8, we know that the derived synchronization operations of our example TGG are deterministic. This allows us to apply Thm. 8.2 and we derive that the derived synchronization framework is correct and complete. □

Invertibility of a synchronization framework intuitively means that the propagation operations are inverse to each other (cf. Def. 3.4). Weak invertibility requires this property for a restricted set of inputs, namely those where the given update on one domain can be interpreted as result of a propagation of an update from the corresponding opposite domain. In order to ensure invertibility, we require additional properties of the TGG. If the source identic triple rules are empty rules on the source and correspondence components and analogously for the target-identic triple rules, then we say that the TGG is *pure*. This condition is used to ensure weak invertibility according to Thm. 8.6 below. In the more specific case that all triple rules of a TGG are creating on the source and target components ($TR = TR^{+s} = TR^{+t}$), then the TGG is called *tight*, because the derived forward and backward rules are strongly related. Effectively, a tight TGG ensures for the operational forward and backward translation rules that each of them changes at least one translation attribute. With other words, for each triple rule *tr* there is a derived forward translation rule $tr_{FT} \in TR^{+s}_{FT}$ and a derived backward translation rule $tr_{BT} \in TR^{+t}_{BT}$. This additional property ensures invertibility according to Thm. 8.6 below.

Definition 8.5 (Pure and Tight TGG) *A TGG is called* pure*, if $TR^{1s} \subseteq TR_T$ and $TR^{1t} \subseteq TR_S$. It is called* tight*, if the sets of source and target creating rules TR^{+s} and TR^{+t} coincide with the set of triple rules TR, i.e., $TR = TR^{+s} = TR^{+t}$.*

Invertibility of the derived synchronization framework means that the propagation operations are inverse to each other, while the notion of weak invertibility requires this property only for a restricted set of inputs (see Def. 3.4). In addition to the conditions for ensuring a correct synchronization framework (Thm. 8.2), the notions of pure and tight TGGs allow us to ensure these properties in Thm. 8.6 below.

Theorem 8.6 (Invertibility and Weak Invertibility) *Let Synch(TGG) be a derived TGG synchronization framework, such that the sets of operational translation rules of TGG are kernel-grounded and deterministic (see Def. 6.5), TGG is pure and at most one set of operational translation rules was extended by a conservative policy, then Synch(TGG) is weakly invertible. If, moreover, TGG is tight and there was no policy applied at all, then Synch(TGG) is also invertible.*

Proof (Idea) To prove weak invertibility law (c1) in Fig. 3.5, we can first show that the intermediate triple graphs after applying bAln, Del and fAln, Del according to Fig. 7.1 and 7.3, are the same in the last two diagrams of (c1). We compute all three diagrams of (c1) and obtain consistency creating sequences via Del for each diagram using the precondition that the operational rules are deterministic (which subsumes termination). Moreover, we derive that the second and the third diagrams contain the same intermediate triple graph G_I. Afterwards, the auxiliary operations fAdd and bAdd for all three diagrams can be executed. We can use the composition and decomposition result for TGGs and the requirements that the TGG is pure, deterministic and preserves functional behavior. If at most one set of operational translation rules are extended by a conservative policy, the proof shows that backward transformation sequences are not eliminated by the policy. This allows us to obtain the resulting diagrams according to law (c1). The proof for axiom (c2) follows out of the symmetry of the definitions. To prove invertibility (laws (d1) and (d2)), we use the preconditions that no policy is applied and that the TGG is tight, i.e., all rules are source and target creating. This ensures that for each forward translation sequence there is a corresponding backward translation sequence. For the full proof see Thm. 1 in [24], where sets of operational rules are called deterministic, if they are kernel-grounded and deterministic using the notions of this article. □

The sets of operational translation rules of *TGG* are kernel-grounded and deterministic according to Fact 7.8 in Sec. 7. Moreover, the TGG is pure and we used the conservative policy for the backward direction only. Thus, Thm. 8.6 ensures that *Synch*(*TGG*) is weakly invertible (see Fact 8.7 below).

Fact 8.7 (Case Study: Weak Invertibility) *The derived synchronization framework for our example TGG is weakly invertible.*

Proof In order to apply Thm. 8.6 concerning weak invertibility, we have to show that the TGG is pure (cf. Sec. 8) and at most one set of operational rules was restricted by a conservative policy (cf. Def. 6.5). The used policy for the set of backward translation rules is conservative, which we have shown already in Fact 8.4. No further policy is applied and the TGG is pure, because each rule is either creating on the source and target component, or it is creating either on the source or the target component and empty on the other components. Therefore, we can apply Thm. 8.6 and derive weak invertibility. □

An intuitive example for weak invertibility is shown in Ex. 3.5 in Sec. 3, where we also show by counterexample that the derived synchronization framework for our example TGG is not invertible in the general sense. The reason is that information about birth dates is stored in one domain only. The automated validation for our example TGG with 8 rules was performed in 25 seconds on a standard consumer notebook via the analysis engine of the tool AGG [37]. We are confident that the scalability of this approach can be significantly improved with additional optimizations.

Remark 8.8 (Applicability of the Approach) We provided sufficient conditions ensuring correctness and completeness (Thm. 8.2) which can be checked statically. In the following, we discuss these restrictions with respect to relevant application scenarios.

1. *Determinism:* Most importantly, we require that the derived sets of operational rules are deterministic, i.e., the forward and backward propagation operations ensure unique results. In several application domains, this property is already a requirement by the domain experts, i.e., has to be ensured anyhow. For example, unique results are often required for the synchronization between visual models and implementation code, i.e., for code generation and reverse engineering. Existing triple rules can be modified to enforce deterministic based on the discussed critical pair analysis of a TGG using the tool AGG. For example, the designer may insert additional correspondence nodes (trace links) to enforce determinism and avoid conflicts between rules. The condition for determinism does not seem to confine the expressiveness of TGG rules. In a large scale industrial project, we used a TGG for the fully automated translation of satellite control software [32], where the used TGG contains more than 200 rules and the derived system of operational rules is deterministic. As a general recommendation based on the experiences from this project, we can state that a designer of a TGG should divide the rules in small groups, such that there are no cyclic dependencies between the groups.
2. *Kernel-grounded sets of operational rules:* Intuitively, the restriction to kernel-grounded rules concerns the possibility that one domain may contain information that is not present in the corresponding opposite domain. When translating from one domain to the other, we apply only those rules that are changing at least one translation attribute (TR^{+s}_{FT} and TR^{+t}_{BT}). Thus, we require that the structures that concern only one domain are handled separately by triple rules that are the identity on the corresponding opposite domain (TR^{1s}_{FT} and TR^{1t}_{BT}) and that these sets of rules do not create structures that may be needed by the first group of rules. This means that the restriction to kernel-grounded sets of operational rules mainly restricts the freedom when designing the TGG and usually not the problem and application domain itself.

The result on invertibility (Thm. 8.6) requires additional properties. Weak invertibility is ensured, if the TGG is pure and at most one of the sets of operational rules is extended by a conservative policy. While this condition is not very restrictive in the experience of the authors, the stronger condition for invertibility requiring a tight TGG practically means that all information in one domain are also reflected in the corresponding opposite domain. This result is consistent with Diskin et al.'s analysis of strong invertibility [9].

In the case that the specified TGG does not ensure deterministic synchronization operations, there are still two options for synchronization that ensure correctness and completeness. On the one hand, the triple rules can be modified in a suitable way, such that the TGG can be verified to be deterministic. For this purpose, the critical pair analysis engine of the tool AGG [37] can be used to analyze conflicts between the generated operational translation rules. Moreover, backtracking can be reduced or even eliminated by generating additional application conditions for the operational translation rules using the automatic generation of filter NACs [22]. On the other hand, the TGG can be used directly, leading to nondeterministic synchronization operations, which may provide several possible synchronization results.

9 Related Work

Triple graph grammars have been successfully applied in multiple case studies for bidirectional model transformation, model integration and synchronization [29,35,15,14], and in the implementation of QVT [19]. Moreover, several formal results are available concerning correctness, completeness, termination [12,16], functional behavior [25,16], and optimization with respect to the efficiency of their execution [22, 30,16]. The presented constructions for performing model transformations and model synchronizations are inspired by Schürr et al. [33,35] and Giese et al. [14,15], respectively. The constructions formalize the main ideas of model synchronization based on TGGs in order to show correctness and completeness of the approach based on the results known for TGG model transformations.

Bidirectional transformation frameworks originate from the lens framework proposed by Foster et al. [13]. Lenses consider the asymmetric synchronization: one model is a view of the other, and define a state-based framework for asymmetric synchronization. "State-based" means that the synchronizer takes the states of models before and after update as input, and produces new states of models as output. Inspired by the lense framework, several researchers propose state-based framework for symmetric synchronization [36, 27,7]. As a more general case, symmetric synchronization allows neither of the model to be a view of the other. However, as Diskin et al. [6] point out, state-based bidirectional transformations actually mix two different operations–delta (correspondence relations between models or between different versions of a model) discovery and delta propagation, leading to several semantic problems. To fix these problems, several researchers [2,6,8,9,28] propose delta-based frameworks, where deltas are taken as input and output. Typical delta-based frameworks include delta lens [6] for the asymmetric cases, and symmetric delta lens [9] and edit lens [28] for the symmetric cases.

The model synchronization framework used in this paper is a simplified version of the symmetric delta lens (sd-lens) framework proposed by Diskin et al. [9]. The difference between this paper and sd-lenses is that we do not consider the weak undoability laws (fUndo) and (bUndo) defined in the sd-lens framework. In addition, Diskin et al. [9] also refine an sd-lens as an alignment framework and a consistency maintainer. Our implementation is consistent with this refinement

as well. The alignment framework corresponds to fAln and bAln operations. The consistency maintainer is implemented by Del, fAdd, and bAdd operations, which first mark the consistent parts of the integrated model, then propagate the changes, and finally delete the remaining inconsistent parts. As a result, this paper also serves as a proof of concept for the theory of symmetric delta lenses.

The BiG system proposed by Hidaka et al. [26] is a bidirectional graph synchronization system. Different from our work based on symmetric TGG specification, the BiG system is based on an unidirectional graph transformation language, UnQL [4], and thus is asymmetric by nature. Accordingly, the BiG system adopts an asymmetric synchronization framework (a variant of the basic lens framework [13]), while our work adopts a simplified version of the symmetric delta lens [9]. In an asymmetric framework, one model has to be a view of the other, and it is not possible to synchronize two models each containing information not presented in the other.

Giese et al. introduced incremental synchronization techniques based on TGGs in order to preserve consistent structures of the given models by revoking previously performed forward propagation steps and their dependent ones [15]. This idea is generalized by the auxiliary operation Del in the present framework, which ensures the preservation of maximal consistent substructures and extends the application of synchronization to TGGs that are not tight or contain rules with negative application conditions. Giese et al. [14] and Greenyer et al. [20] proposed to extend the preservation of substructures by allowing for the reuse of any *partial* substructure of a rule causing, however, non-deterministic behavior. However, a partial reuse can cause unintended results. Consider, e.g., the deletion of a person A in the source domain and the addition of a new person with the same name, then the old birth date of person A could be reused.

In order to improve efficiency, Giese et al. [15,14] proposed to avoid the computation of already consistent substructures by encoding the matches and dependencies of rule applications within the correspondences. In the present framework, operation Del can be extended conservatively by storing the matches and dependency information separately, such that the provided correctness and completeness results can be preserved as presented in Sec. 8.

Becker et al. presented a generally non-deterministic synchronization approach based on TGGs [3] using the PROGRES approach [34] with the focus to integration, i.e., construction of missing correspondence links. The algorithm requires user interaction at each rule application, where some integration rules are in conflict for partial matches. For general TGGs, such integrations may require backtracking to achieve a resulting model that is fully integrated. In principle, it might be possible to adapt this algorithm in order to apply the main results in this article on correctness and completeness, since the actual steps are performed via the operational rules of a TGG.

10 Conclusion

Based on our formal framework for correctness, completeness, termination and functional behavior of model transformations using triple graph grammars (TGGs) [12,22], we presented a formal TGG framework for model synchronization inspired by [15,14,33,35]. The main results (Thms. 8.2 and 8.6) show correctness, completeness and (weak) invertibility, provided that the derived synchronization operations are deterministic. Based on general results for TGGs in [22], Thm. 8.1 and Sec. 6 provide sufficient static conditions for checking that the operations are deterministic.

However, if the operations are not yet deterministic, we may be able to define a conservative policy in order to ensure determinism (see Ex. 8.3). Note that our notion of correctness in Def. 3.3 requires that the synchronization operations are deterministic. But if we drop this requirement for correctness we are optimistic that our theory can be extended to handle also applications based on TGGs with nondeterministic model transformations. In fact, the theory of TGGs in general is not restricted to the deterministic case.

In future work, the Henshin tool [1] based on AGG [37] will be extended in order to implement the synchronization algorithm for forward propagation in Fig. 7.3. The implementation will also support concurrent model synchronization with conflict resolution based on our approach in [21], where we extended the synchronization framework of this article to the concurrent case and we plan to apply these techniques within an industrial research project [32]. Furthermore, we will study model synchronization based on non-deterministic forward and backward propagation operations in more detail. Finally, the relationship with lenses [36] and delta-based bidirectional transformations [9] will be studied in more detail, especially in respect of the composition of lenses leading to the composition of synchronization operations. It is also interesting to investigate the potential of TGG's to provide a common implementation framework for a family of algebraic models for different synchronization modes described in [8].

11 Acknowledgment

This work was partially supported by

- the Ontario Research Fond's Research Excellence Project on Model-Integrated Software Service Engineering,
- the National Natural Science Foundation of China under Grant No. 61202071 and No. 61121063.,
- the CICYT project (ref. TIN2007-66523) and by the AGAUR grant to the research group ALBCOM (ref. 00516) and
- the Fonds National de la Recherche, Luxembourg (3968135).

Fonds National de la Recherche Luxembourg

References

1. Arendt, T., Biermann, E., Jurack, S., Krause, C., Taentzer, G.: Henshin: Advanced concepts and tools for in-place EMF model transformations. In: Petriu, D.C., Rouquette, N., Haugen, Ø. (eds.) Proc. of the ACM/IEEE 13th Intern. Conf. on Model Driven Engineering Languages and Systems (MoDELS'10). LNCS, vol. 6394, pp. 121–135. Springer (2010)
2. Barbosa, D.M., Cretin, J., Foster, N., Greenberg, M., Pierce, B.C.: Matching lenses: alignment and view update. SIGPLAN Not. 45(9), 193–204 (Sep 2010), http://doi.acm.org/10.1145/1932681.1863572
3. Becker, S., Nagl, M., Westfechtel, B.: Incremental and interactive integrator tools for design product consistency. In: Nagl, M., Marquardt, W. (eds.) Collaborative and Distributed Chemical Engineering. From Understanding to Substantial Design Process Support, LNCS, vol. 4970, pp. 224–267. Springer Berlin / Heidelberg (2008), http://dx.doi.org/10.1007/978-3-540-70552-9_11
4. Buneman, P., Fernandez, M., Suciu, D.: UnQL: a query language and algebra for semistructured data based on structural recursion. The VLDB Journal 9(1), 76–110 (2000)
5. Czarnecki, K., Foster, J., Hu, Z., Lämmel, R., Schürr, A., Terwilliger, J.: Bidirectional Transformations: A Cross-Discipline Perspective. In: Proc. ICMT'09. LNCS, vol. 5563, pp. 260–283. Springer (2009)
6. Diskin, Z., Xiong, Y., Czarnecki, K.: From State- to Delta-Based Bidirectional Model Transformations: the Asymmetric Case. Journal of Object technology 10, 6:1–25 (2011)
7. Diskin, Z.: Algebraic Models for Bidirectional Model Synchronization. In: Czarnecki, K., Ober, I., Bruel, J.M., Uhl, A., Vlter, M. (eds.) Model Driven Engineering Languages and Systems, Lecture Notes in Computer Science, vol. 5301, pp. 21–36. Springer Berlin / Heidelberg (2008), http://dx.doi.org/10.1007/978-3-540-87875-9_2, 10.1007/978-3-540-87875-9_2
8. Diskin, Z.: Model Synchronization: Mappings, Tiles, and Categories. In: Generative and Transformational Techniques in Software Engineering III, LNCS, vol. 6491, pp. 92–165. Springer (2011)
9. Diskin, Z., Xiong, Y., Czarnecki, K., Ehrig, H., Hermann, F., Orejas, F.: From State- to Delta-based Bidirectional Model Transformations: The Symmetric Case. In: Proc. MODELS 2011. Springer (2011)
10. Ehrig, H., Ehrig, K., Hermann, F.: From Model Transformation to Model Integration based on the Algebraic Approach to Triple Graph Grammars. EC-EASST 10 (2008)
11. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. EATCS Monographs in Theor. Comp. Science, Springer (2006)
12. Ehrig, H., Ermel, C., Hermann, F., Prange, U.: On-the-Fly Construction, Correctness and Completeness of Model Transformations based on Triple Graph Grammars. In: Proc. MODELS'09. LNCS, vol. 5795, pp. 241–255. Springer (2009)
13. Foster, J.N., Greenwald, M.B., Moore, J.T., Pierce, B.C., Schmitt, A.: Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. ACM Trans. Program. Lang. Syst. 29(3) (May 2007), http://doi.acm.org/10.1145/1232420.1232424
14. Giese, H., Hildebrandt, S.: Efficient Model Synchronization of Large-Scale Models . Tech. Rep. 28, Hasso Plattner Institute at the University of Potsdam (2009)
15. Giese, H., Wagner, R.: From model transformation to incremental bidirectional model synchronization. Software and Systems Modeling 8(1), 21–43 (2009)
16. Giese, H., Hildebrandt, S., Lambers, L.: Toward Bridging the Gap Between Formal Semantics and Implementation of Triple Graph Grammars . Tech. Rep. 37, Hasso Plattner Institute at the University of Potsdam (2010)
17. Golas, U., Ehrig, H., Habel, A.: Multi-amalgamation in adhesive categories. In: Proceedings of Intern. Conf. on Graph Transformation (ICGT' 10) (2010), http://tfs.cs.tu-berlin.de/publikationen/Papers10/GEH10.pdf, to appear
18. Golas, U., Ehrig, H., Hermann, F.: Formal Specification of Model Transformations by Triple Graph Grammars with Application Conditions. EC-EASST 39 (2011)
19. Greenyer, J., Kindler, E.: Comparing relational model transformation technologies: implementing query/view/transformation with triple graph grammars. Software and Systems Modeling (SoSyM) 9(1), 21–46 (2010)
20. Greenyer, J., Pook, S., Rieke, J.: Preventing information loss in incremental model synchronization by reusing elements. In: Proc. ECMFA 2011. LNCS, vol. 6698, pp. 144–159. Springer Verlag (2011)
21. Hermann, F., Ehrig, H., Ermel, C., Orejas, F.: Concurrent Model Synchronization with Conflict Resolution Based on Triple Graph Grammars. In: de Lara, J., Zisman, A. (eds.) Proc. Fundamental Aspects of Software Engineering (FASE'12). LNCS, vol. 7212, pp. 178–193. Springer (2012)
22. Hermann, F., Ehrig, H., Golas, U., Orejas, F.: Efficient Analysis and Execution of Correct and Complete Model Transformations Based on Triple Graph Grammars. In: Proc. MDI'10 (2010)
23. Hermann, F., Ehrig, H., Orejas, F., Czarnecki, K., Diskin, Z., Xiong, Y.: Correctness of model synchronization based on triple graph grammars. In: Proc. Int. Conf. on Model Driven Engineering Languages and Systems (MoDELS'11). LNCS, vol. 6981. Springer (2011)
24. Hermann, F., Ehrig, H., Orejas, F., Czarnecki, K., Diskin, Z., Xiong, Y.: Correctness of Model Synchronization Based on Triple Graph Grammars - Extended Version. Tech. Rep. TR 2011-07, TU Berlin, Fak. IV (2011)
25. Hermann, F., Ehrig, H., Orejas, F., Golas, U.: Formal Analysis of Functional Behaviour of Model Transformations Based on Triple Graph Grammars. In: Proc. ICGT'10. LNCS, vol. 6372, pp. 155–170. Springer (2010)
26. Hidaka, S., Hu, Z., Inaba, K., Kato, H., Matsuda, K., Nakano, K.: Bidirectionalizing Graph Transformations. In: 15th ACM SIGPLAN International Conference on Functional Programming (ICFP 2010). pp. 205–216 (2010)
27. Hofmann, M., Pierce, B., Wagner, D.: Symmetric lenses. SIGPLAN Not. 46(1), 371–384 (Jan 2011), http://doi.acm.org/10.1145/1925844.1926428
28. Hofmann, M., Pierce, B., Wagner, D.: Edit lenses. SIGPLAN Not. 47(1), 495–508 (Jan 2012), http://doi.acm.org/10.1145/2103621.2103715
29. Kindler, E., Wagner, R.: Triple graph grammars: Concepts, extensions, implementations, and application scenarios. Tech. Rep. TR-ri-07-284, Department of Computer Science, University of Paderborn, Germany (2007)
30. Klar, F., Lauder, M., Königs, A., Schürr, A.: Extended Triple Graph Grammars with Efficient and Compatible Graph Translators. In: Graph Transformations and Model Driven Enginering, LNCS, vol. 5765, pp. 141–174. Springer Verlag (2010)

31. Object Management Group: Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. Version 1.0 formal/08-04-03. http://www.omg.org/spec/QVT/1.0/ (2008)
32. Ottersten, B., Engel, T.: SPELL: Satellite communications 'lingua franca, in: Interdisciplinary Centre for Security, Reliability and Trust - Annual Report 2011. pp. 14–15. scienceRELATIONS, Dortmund/Berlin, Germany (2012), http://www.uni.lu/content/download/52106/624943/version/1/file/SnT_AR2011_final_web.pdf
33. Schürr, A.: Specification of Graph Translators with Triple Graph Grammars. In: Proc. WG'94. LNCS, vol. 903, pp. 151–163. Springer Verlag, Heidelberg (1994)
34. Schürr, A., Winter, A., Zündorf, A.: The PROGRES approach: Language and environment. In: Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G. (eds.) Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages and Tools. p. 487550. World Scientific (1999)
35. Schürr, A., Klar, F.: 15 Years of Triple Graph Grammars. In: Proc. ICGT'08. LNCS, vol. 5214, pp. 411–425 (2008)
36. Stevens, P.: Bidirectional Model Transformations in QVT: Semantic Issues and Open Questions . Software and Systems Modeling 9, 7–20 (2010)
37. TFS-Group, TU Berlin: AGG (2011), http://tfs.cs.tu-berlin.de/agg