Towards Domain Completeness for Model Transformations Based on Triple Graph Grammars

Nico Nachtigall[1], Frank Hermann[1], Benjamin Braatz[1], and Thomas Engel[1]

Interdisciplinary Centre for Security, Reliability and Trust,
Université du Luxembourg, Luxembourg
firstname.lastname@uni.lu

Abstract. The analysis of model transformations is a challenging research area within model driven engineering. Triple graph grammars (TGGs) have been applied in various transformation scenarios and their formal foundation has been a vital ground for general results concerning notions of correctness and completeness.
This paper addresses existing gaps between practical scenarios and the formal results of TGGs concerning the notion of completeness. Since the source domain language of a model transformation is usually specified independently from the TGG, we use the notion of domain completeness, which requires that the model transformation has to provide a corresponding target model for each model of the source domain language. As main result, we provide a general method for showing that the source domain language is included in the language that is generated by the source rules of the TGG. This provides the first of two components for verifying domain completeness. The running example is the well studied object-relational mapping.

Keywords: model transformation, completeness, graph grammars, constraints

1 Introduction

Triple graph grammars (TGGs) [1,2,3] are a well-established concept for the specification and execution of bidirectional model transformations within model driven software engineering. Their main advantage is an automatic generation of operational rules for forward and backward model transformations, which simplifies specification and enhances usability as well as consistency. Several formal results for analysing general properties of model transformations based on TGGs have been developed. In present work [4], the notion of completeness and correctness of model transformations via TGGs is based on the language $\mathcal{L}(TGG)$ that is induced by the TGG itself.

However, in practical scenarios, the source and target languages are given independently from the TGG. We consider the general case where the source domain language $\mathcal{L}_S = \mathcal{L}(TG_S, C_S)$ is given by a source type graph TG_S and a set of source constraints C_S. The TGG generates the language $\mathcal{L}(TGG)^{\mathrm{S}}$ of source models. In this more general case, we may observe that $\mathcal{L}_S \neq \mathcal{L}(TGG)^{\mathrm{S}}$. This is not problematic, as long as we can still ensure completeness with respect to $\mathcal{L}_S$, which we call domain-completeness in this paper.

Definition 1 (Domain Completeness). *A model transformation MT with source language $\mathcal{L}_S$ is called* domain complete*, if it generates a target model M_T for each source model $M_S \in \mathcal{L}_S$.* $\triangle$

The challenge for showing domain completeness is to verify that $\mathcal{L}_S \subseteq \mathcal{L}(TGG)^{\mathrm{S}}$. The main contribution of this paper provides the first of two steps for showing this property. We provide a general technique for verifying that a language defined by a type graph and constraints is contained in a language that is defined using a non-deleting graph grammar. Since, TGGs consist of non-deleting rules only, we can instantiate this result for showing that the source domain language is contained in the language that is generated by the source rules of the TGG, i.e. $\mathcal{L}_S \subseteq \mathcal{L}(TGG_S)$. The second step is then to show that $\mathcal{L}(TGG_S) \subseteq \mathcal{L}(TGG)^S$ using the constraints of $\mathcal{L}_S$.

In Sec. 2, we present the general scenario and framework for proving domain completeness and Sec. 3 presents our method and main result for showing that a language defined by constraints is contained in a language defined by a non-deleting grammar. Sec. 4 discusses related work and Sec. 5 summarises the main results and describes aspects of future work.

2 General Framework

In the general case of model transformations $MT\colon \mathcal{L}_S \Rrightarrow \mathcal{L}_T$ between domain specific languages (DSLs) $\mathcal{L}_S$ and $\mathcal{L}_T$, we cannot assume that the languages are specified with graph grammars. In many application scenarios, DSLs are specified by a meta model and OCL constraints [5]. In the theory of graph transformation systems, meta models correspond to type graphs and constraints to nested graph constraints [6]. Hence, we generally assume that the source language is given by $\mathcal{L}_S = (TG_S, C_S)$ with type graph TG_S and constraints C_S.

Graph constraints and grammars allow the definition of graph languages in a declarative, or respectively, procedural manner. In this paper, we use the general notion of $\mathcal{M}$-adhesive transformation systems [7,8] as basis for plain graph grammars GG and triple graph grammars TGG.

$$\begin{tikzcd} L \arrow[d, "m"'] \arrow[dr, phantom, "(PO)"] & K \arrow[l] \arrow[r] \arrow[d] \arrow[dr, phantom, "(PO)"] & R \arrow[d] \\ G & D \arrow[l] \arrow[r] & H \end{tikzcd}$$

A transformation rule p is given by a span $p = (L \leftarrow K \rightarrow R)$ of injective graph-morphisms (mappings for nodes and edges) with left hand side L and right hand side R. A transformation step $G \xRightarrow{p,m} H$ via match morphism $m\colon L \rightarrow G$ is given by two pushouts as depicted on the right. Intuitively, H is obtained by removing from G the parts in $L \setminus K$ at $m(L)$ and adding $R \setminus K$. In the case of a non-deleting rule $p = (L \xleftarrow{id} L \rightarrow R)$ the rule is simply $p = (L \rightarrow R)$ and the first pushout is ommitted.

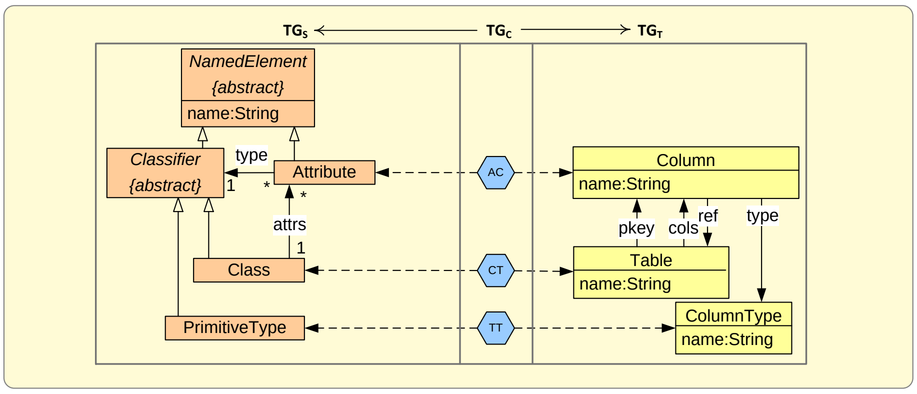


Fig. 1. Domain meta-model (TG_S) and triple type graph ($TG_S \leftarrow TG_C \rightarrow TG_T$)

A triple graph $G = (G^{\mathrm{S}} \xleftarrow{s_G} G^{\mathrm{C}} \xrightarrow{t_G} G^{\mathrm{T}})$ is an integrated model consisting of a source graph G^{S}, a target graph G^{T} and explicit correspondences given by correspondence graph G^{C} together with graph morphisms $s_G : G^{\mathrm{C}} \to G^{\mathrm{S}}$ and $t_G : G^{\mathrm{C}} \to G^{\mathrm{T}}$.

As running example, we use a variant of the well-known model transformation from class diagrams to relational database models (CD2RDBM) [9,2].

Example 2 (Meta model and triple type graph). Fig. 1 depicts the meta-model TG_S of the source language and the type graph $TG_S \leftarrow TG_C \rightarrow TG_T$ of the TGG. Class diagrams contain classes (Class) with Attributes (Attribute) and relational database models contain corresponding tables (Table) with columns (Column). Abstract node types (label {abstract}) may not appear in instances and multiplicity constraints on edge types type and attrs require that each Attribute has exactly one type and is contained in exactly one Class. △

Formally, we use graph constraints in the notion of (nested) conditions according to [6] providing the concepts for both, graph constraints and application conditions for rules. Conditions consist of formulas in first order logic over morphisms (implications) $P \to C$ with premise graph P and conclusion graph C.

Remark 3 (Matches and instances of constraints $\mathrm{Inst}(C)$*).* In the context of model transformations, rules should be applied along match morphisms that do not identify structures of graphs, but which may identify attribute expressions to identical values. This class of morphisms is called *almost injective* [10]. Moreover, matches may map nodes to nodes with a more concrete type (type refinement) according to the inheritance relation in the type graph (cf. clan morphisms [8,11]). The same situation arises for matches of constraints. Therefore, we use the concept of a schema constraint, which interprets a constraint c as the disjunction of its possible instances $\mathrm{Inst}(c)$ which may occur in a graph. The instances $\mathrm{Inst}(C)$ of conditions C subsume all possible type refinements and merges of data values along match morphism from conditions in C. △

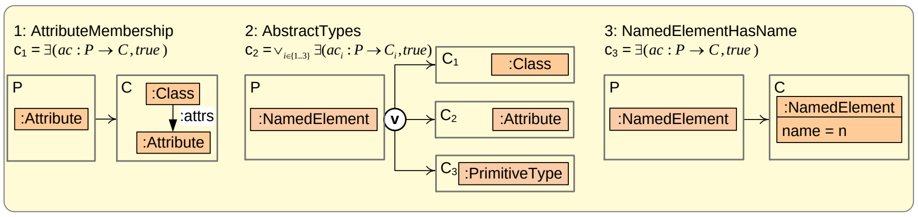


Fig. 2. Some domain constraints C_S of domain language $\mathcal{L}_S$

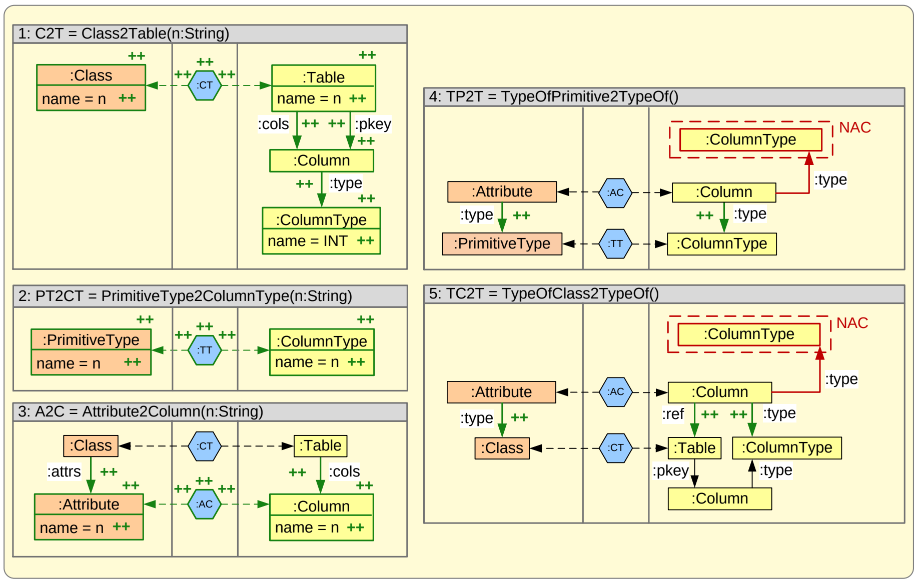


Fig. 3. Triple rules of TGG for CD2RDBM transformation

Example 4 (Graph Constraints). Three of nine domain constraints C_S for $\mathcal{L}_S$ are depicted in Fig. 2. They subsume requirements concerning multiplicities (constraint 1 – each attribute must be contained in a class) and forbidden abstract types (constraint 2 – each named element must be of type Class, Attribute or PrimitiveType) given in Fig. 1. Additionally, each named element must have one name (constraint 3). △

$$\begin{tikzcd} G^{\mathrm{S}} \arrow[d, "m^{\mathrm{S}}"'] & G^{\mathrm{C}} \arrow[l] \arrow[r] \arrow[d, "m^{\mathrm{C}}"'] \arrow[l, phantom, "="] \arrow[r, phantom, "="] & G^{\mathrm{T}} \arrow[d, "m^{\mathrm{T}}"] \\ H^{\mathrm{S}} & H^{\mathrm{C}} \arrow[l] \arrow[r] & H^{\mathrm{T}} \end{tikzcd}$$

Triple graphs are related by triple graph morphisms $m = (m^{\mathrm{S}}, m^{\mathrm{C}}, m^{\mathrm{T}}) : G \rightarrow H$ [1,2] consisting of three graph morphisms that preserve the associated correspondences (i.e., the diagrams on the right commute). A triple rule tr is given by a morphism ($tr \colon L \rightarrow R$). Thus, it is non-deleting and specifies how a given consistently integrated model can be extended simultaneously on all three components yielding again a consistently integrated model. Moreover, triple rules can be extended by application conditions for restricting their application to specific matches [10].

Example 5 (Triple Rules). The integrated models of our running example are specified by the triple rules in Fig. 3. Rule 1 (C2T) creates a Class with its corresponding Table having the same name and a column of type INT that stores the primary key (edge pkey). Rule 2 (PT2CT) creates a primitive type (PrimitiveType) with its mapped ColumnType. Rule 3 (A2C) creates an Attribute as a class member (edge attrs) with its mapped Column that belongs to the corresponding Table (edge cols). Rule 4 (TP2T) connects an Attribute with a PrimitiveType as type and correspondingly, connects a Column via edge type with the ColumnType corresponding to the PrimitiveType, but only, if no other ColumnType is already defined for this Column (negative application condition NAC). Rule 5 (TC2T) connects an Attribute with a class as type and correspondingly, connects a Column via edge ref with the Table corresponding to the Class and via edge type with the corresponding ColumnType determined by the primary key, but only, if no other ColumnType is already defined for this Column. △

A triple graph grammar $TGG = (TG, SG, P)$ consists of a triple type graph TG, a triple start graph SG and a set P of triple rules, and generates the triple graph language of consistently integrated models $\mathcal{L}(TGG) \subseteq \mathcal{L}(TG)$ with consistent source and target languages $\mathcal{L}(TGG)^{\mathrm{S}} = \{G^{\mathrm{S}} \mid (G^{\mathrm{S}} \leftarrow G^{\mathrm{C}} \rightarrow G^{\mathrm{T}}) \in \mathcal{L}(TGG)\}$ and $\mathcal{L}(TGG)^{\mathrm{T}} = \{G^{\mathrm{T}} \mid (G^{\mathrm{S}} \leftarrow G^{\mathrm{C}} \rightarrow G^{\mathrm{T}}) \in \mathcal{L}(TGG)\}$. The operational rules of a TGG for forward and backward model transformations are derived by an automatic construction [3,10]. The operational source rules TGG_S of a TGG are obtained by restricting the rules of P to their source components.

3 C-Extensions for Domain Completeness

This section presents the first of two parts for showing domain completeness of a model transformation via TGGs, namely showing that a given source language $\mathcal{L}_S = \mathcal{L}(TG_S, C_S)$ with constraints C_S is contained in the language $\mathcal{L}'_S = \mathcal{L}(TG_S, TGG_S)$ generated by the grammar of source rules TGG_S ($\mathcal{L}_S \subseteq \mathcal{L}'_S$). Effectively, this means to provide a method for showing that all graphs satisfying the constraints can be constructed via the source rules. We introduce the general notion of C-extension completeness of a language for verifying language inclusions. This notion instantiates to the special case of non-deleting grammars of source rules in our TGG scenario. C-extension completeness is used to ensure that the constraints guarantee all language restrictions that are induced by the grammar. We iterate over all minimal fragments of the meta model and – when needed – extend them to show that they can be constructed via the grammar. For full formal details we refer to [12].

Def. 6 defines the step-wise extension of a graph G via constraints C. A constraint c is *violation stable under embedding*, if for any graph G that violates c it holds that for any inclusion $G \hookrightarrow H$ also H violates c. Thus, we can neglect graphs that do not fulfill violation stable constraints (namely C-inconsistent graphs) when computing extensions.

The extension of a graph G via a set of constraints C is defined recursively starting with the initial extension that contains graph G only. A new extension is derived from an existing extension E as follows: *a*) Let G_E be a graph of E, let c be an instance of a constraint in C without negations that may have one or more conclusions connected by disjunctions and let $m\colon P \to G_E$ be a match from the premise P of c to G_E. *b*) Compute all overlappings of the conclusions of c with G_E with respect to m. *c*) For each overlapping, a new graph G'_E is potentially added to E leading to extension E' by adding the non-overlapping part of the conclusion to G_E with extension step $E \xRightarrow{extend(G_E,c)} E'$.

$$\begin{tikzcd}
P \arrow[r, "p"] \arrow[rr, bend left, "f"] \arrow[dr, "m"'] & P' \arrow[r, "f'"] \arrow[d, "m'"'] \arrow[dr, phantom, "(1)"] & C \arrow[d] \\
 & G_E \arrow[r] & G'_E
\end{tikzcd}$$

Definition 6 (C-Extensions). *Let G be a graph. The* extensions of G via morphism f and match m *form the set of graphs given by $extend(G, f, m)$ below. The* extensions of G via a constraint c and a match m *form the set of graphs given by $extend(G, c, m)$ below. The* extensions of G via a set of constraints C *form the set of sets of graphs given by $Extensions(G, C)$ below.*

– $extend(G_E, f, m) = \{G'_E \mid$ (1) above is a pushout with $f, f', m, m', p \in \mathcal{M}$, $m' \circ p = m$, and $f' \circ p = f$, G'_E is not C-inconsistent$\}$

– $extend(G_E, c, m) = \bigcup_{i\in I} extend(G_E, ac_i, m), m \in \mathcal{M}$, $c \equiv \vee_{i\in I}\exists(ac_i\colon P \to C_i, \boldsymbol{true})$

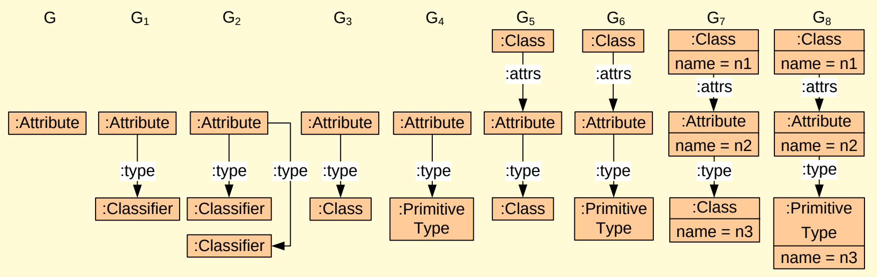
– $Extensions(G, C) = \{\{G\}\} \cup \{E' \mid E' = E \setminus \{G_E\} \cup extend(G_E, c, m)$, $E \in Extensions(G, C), G_E \in E, c \in \mathrm{Inst}(C)$, $c \equiv \vee_{i\in I}\exists(ac_i\colon P \to C_i, \boldsymbol{true}), m\colon P \to G_E \in \mathcal{M}\}$ △

In practice, C-extensions are considered only up to isomorphism.

Example 7 (C-Extensions). Fig. 4 depicts some extensions $Extensions(G, C_S)$ of graph G via constraints C_S of Fig. 2. The extensions are obtained by the following extension steps: $\{G\} \xRightarrow{extend(G,c_5)} \{G_1\} \xRightarrow{extend(G_1,c_5)} \{G_1, G_2\}$; $\{G_1\} \xRightarrow{extend(G_1,c_2)} \{G_3, G_4\} \xRightarrow{extend(G_3,c_1)^*} \{G_4, G_5\} \xRightarrow{extend(G_4,c_1)} \{G_5, G_6\} \xRightarrow{extend(G_5,c_3)^*} \{G_6, G_7\} \xRightarrow{extend(G_6,c_3)^*} \{G_7, G_8\}$. Constraint c_5 is similar to c_1 and ensures that each attribute is of type Classifier. △

For C-extension completeness it is sufficient to consider only the smallest graphs that may occur in a language, namely effective atoms, and from which more complex graphs can be composed. With $Atoms(ATG)$ we denote the set of those graphs that are typed over an attributed type graph ATG and that are atomic in the sense that they can not be divided into smaller subgraphs. Therefore, for attributed graphs the structure of each atom is given by either *a*) an empty graph, or *b*) a node, or *c*) an edge with source and target nodes, or *d*) an attribute edge with source and target.

With $EAtoms(\mathcal{L})$ we denote the set of effective atoms of a language $\mathcal{L}$ that is typed over ATG. Effective atoms are those atoms in $Atoms(ATG)$ that may occur in graphs of language $\mathcal{L}$.



$$Extensions(G, C_S) = \{\{G\}, \{G_1\}, \{G_1, G_2\}, \{G_3, G_4\}, \{G_4, G_5\}, \{G_5, G_6\}, \{G_6, G_7\}, \{G_7, G_8\}, \dots\}.$$

Fig. 4. C-extensions of effective atom : Attribute

Example 8 (Effective Atom). The effective atoms with respect to the domain language $\mathcal{L}_S = \mathcal{L}(TG_S, C_S)$ of the running example are those atoms in $Atoms(TG_S)$ that fulfill the domain constraint 2 for abstract types in Fig. 2. Graphs G, G_3 and G_4 in Fig. 4 are effective atoms with respect to $\mathcal{L}_S$. Graph G_1 is an atom but not effective, since, Classifier is an abstract type. Graphs G_2, G_5, G_6, G_7 and G_8 are not atoms. △

C-extension completeness (cf. Def. 9) of a language $\mathcal{L}$ typed over ATG with respect to a set of constraints C states that for all effective atoms over ATG, an extension via constraints C can be found that is in $\mathcal{L}$.

Definition 9 (C-Extension Completeness). *Let C be a set of constraints typed over ATG. Then, a language $\mathcal{L}$ typed over ATG is called C-*extension complete*, if* $\forall\ a \in EAtoms(ATG) \colon \exists\ S \in Extensions(a, C) \colon S \subseteq \mathcal{L}$. △

Example 10 (C-Extension Completeness). We show C_S-extension completeness of language $\mathcal{L}(TG_S, TGG_S)$ from Sec. 2 (cf. Fig. 1 for TG_S, Fig. 2 for C_S and Fig. 3 for TGG_S). For each effective atom over TG_S an extension via constraints C_S must be found that is contained in language $\mathcal{L}(TG_S, TGG_S)$. An extension is contained in the language, if it can be constructed via the rules in TGG_S. For atom G in Fig. 4, the extension $\{G_7, G_8\} \in Extensions(G, C_S)$ can be found via C_S (cf. Ex. 7). The extension can be constructed by applying source rules $(1, 1, 3, 5)$ or $(1, 2, 3, 4)$ successively leading to graphs G_7 or G_8, respectively. △

In order to ensure termination, when checking C-extension completeness one can define an upper bound for the graph size of input graphs for the model transformation. In addition to C-extension completeness another property called C-conflict-freeness of marking rules is neccessary in order to verify full language inclusions $\mathcal{L}_1 = \mathcal{L}(TG, C) \subseteq \mathcal{L}_2 = \mathcal{L}(TG, GG)$ of languages $\mathcal{L}_1$ and $\mathcal{L}_2$. For a non-deleting grammar, the set of marking rules contains for each rule r, a marking rule r'. Whenever r creates an element x (node, edge or attribute), then r' preserves this element and updates its marker from **F** (false) to **T** (true).

The conflict-freeness of the marking rules with respect to a language $\mathcal{L}$ with constraints C is analysed by performing a critical pair analysis with AGG [13]. Similarly to C-inconsistent graphs, a critical pair $(K_1 \Leftarrow O \Rightarrow K_2)$ is C-inconsistent if graph O violates a violation stable constraint of constraints C. C-inconsistent critical pairs do not need to be analysed, since, there exists a violation stable constraint that forbids the critical pair and any of its embeddings into larger contexts to be in $\mathcal{L}$. This leads to the notion of *C-conflict-freeness* of marking rules. The marking rules are C-conflict-free, if for each critical pair $K_1 \xLeftarrow{p_1,o_1} O \xRightarrow{p_2,o_2} K_2$ that is not C-inconsistent with marking rules p_1 and p_2, the rules and matches are the same $(p_1 = p_2, o_1 = o_2)$.

The main result is stated by Thm. 11. Intuitively, the inclusion holds if each graph G in $\mathcal{L}_1$ can be decomposed into its atoms $G' \subseteq G$ such that for each atom G' an extension $E \subseteq G$ can be constructed via constraints C that is contained in $\mathcal{L}_2$ and the composition of the extensions leads to graph G in $\mathcal{L}_2$ again by applying the rules of grammar GG.

Theorem 11 (C-extension Completeness). *Let $\mathcal{L}_1 = \mathcal{L}(ATG, C)$ be a language typed over ATG and with constraints C and let language $\mathcal{L}_2 = \mathcal{L}(ATG, GG)$ be restricted by a non-deleting grammar $GG = (ATG, SG, P)$ with an empty start graph $SG = \emptyset$. If the marking rules $m(GG)$ are C-conflict-free and $\mathcal{L}_2$ is C-extension complete, then $\mathcal{L}_1 \subseteq \mathcal{L}_2$.* △

Proof (Idea). Let $G \in \mathcal{L}_1$. G can be decomposed into its atoms $A = Atoms(G)$ with $G = \bigcup_{a \in A}(a)$. C-extension completeness of $\mathcal{L}_2$ ensures that each atom a can be extended via C to a_E with $a_E \in \mathcal{L}_2$ and $a_E \subseteq G$. Therefore, a_E can be created via GG. By the equivalence of marking and transformation sequences, each a_E can be fully marked with *true*. The C-conflict freeness of the marking rules allows to apply the Local-Church-Rosser-Theorem and we derive a marking sequence that fully marks all extended atoms a_E to *true*. Thus, there is a sequence via GG that creates G $(G \in \mathcal{L}_2)$. The full proof is given in [12]. □

For the running example in Sec. 2 we successfully verified the language inclusion $\mathcal{L}_1 = \mathcal{L}(TG_S, C_S) \subseteq \mathcal{L}_2 = \mathcal{L}(TG_S, TGG_S)$. Language $\mathcal{L}_2$ is C_S-extension complete and the marking rules $m(TGG_S)$ are C_S-conflict free. This means that the domain constraints C_S are strict enough to cover all language restrictions that are induced by the source rules in TGG_S.

4 Related Work

The formal construction and analysis of model transformations based on TGGs has been started in [2] by analysing information preservation of bidirectional model transformations and continued in a series of papers concerning correctness and completeness, e.g. [14,15]. Pattern-based model-to-model transformations have been introduced in [16] and show a strong correspondence to TGGs. Corresponding correctness and completeness and termination results have been presented in [17]. The existing results, however, do not concern the actual source domain language, whose specification is independent from the TGG and given by the application scenario for the model transformation. In [18], the notion of total TGGs is introduced similar to the notion of domain completeness. A TGG is total if it generates a valid target model for each valid source domain model where validity is defined by conformance with meta-models and the satisfaction of domain constraints. Totality is checked by analysing OCL invariants that must hold for pairs of source and target models.

The concept of translation attributes [10] is applied in this paper for constructing marking rules that allowed us to apply the theory for critical pair analysis in the context of domain completeness. Translation attributes were inspired by the translation algorithm in [3], which uses a set for storing the elements that have been translated during a transformation.

In [2], a similar case study based on forward rules is presented, but without using NACs. The grammar with NACs in this paper handles primary keys and foreign keys in a more appropriate way and allows us to illustrate the formal details and possible differences between the involved language types.

5 Conclusion & Discussion

In formal analysis of model transformations, completeness is certainly one of the most important properties in many domains of model driven engineering. In this paper, we described the general scenario in which model transformations based on triple graph grammars (TGGs) are used. We introduced the notion of domain-completeness, in order to be general enough to match practical scenarios, where domain languages are specified by meta models and constraints. If the model transformation does not concern the complete source domain, we assume that the source language $\mathcal{L}_S$ specifies the relevant part.

In most previous results it was required that the source domain language is contained in the language that is derived by restricting the integrated language generated by the TGG to the source component. This paper closes one half of this gap by providing a method for showing that the source domain language is contained in the language that is generated by the source rules of the TGG. For this purpose, we provided general results for graph transformation systems and extended the existing formal results for TGGs.

The restriction of the results to non-deleting grammars seems to be very strict but fits perfectly to the non-deleting nature of TGGs for model transformations. However, an extension to other (possibly deleting) model transformation formalisms is interesting and left for future work. Furthermore, we will extend the method by the second step to show domain completeness and to apply the presented approach to a model transformation in the domain of satellite control languages.

Acknowledgements We thank Barbara König and Hartmut Ehrig for their comments on preparing this paper.

Supported by the Fonds National de la Recherche, Luxembourg (4895603).

Fonds National de la Recherche Luxembourg

References

1. Schürr, A.: Specification of graph translators with triple graph grammars. In: Graph-Theoretic Concepts in Computer Science. Volume 903 of LNCS., Springer (1994) 151–163
2. Ehrig, H., Ehrig, K., Ermel, C., Hermann, F., Taentzer, G.: Information preserving bidirectional model transformations. In: Fundamental Approaches to Software Engineering. Volume 4422 of LNCS., Springer (2007) 72–86
3. Schürr, A., Klar, F.: 15 Years of Triple Graph Grammars. In: Proc. ICGT'08. Volume 5214 of LNCS. (2008) 411–425
4. Hermann, F., Ehrig, H., Orejas, F., Czarnecki, K., Diskin, Z., Xiong, Y., Gottmann, S., Engel, T.: Model synchronization based on triple graph grammars: correctness, completeness and invertibility. Software & Systems Modeling (2013) 1–29
5. Object Management Group: Object Constraint Language, Version 2.2. (2010)
6. Habel, A., Pennemann, K.H.: Correctness of high-level transformation systems relative to nested conditions. Math. Struct. in Computer Science **19** (2009) 1–52
7. Ehrig, H., Golas, U., Hermann, F.: Categorical Frameworks for Graph Transformation and HLR Systems based on the DPO Approach. Bulletin of the EATCS **102** (2010) 111–121
8. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Springer (2006)
9. Bezivin, J., Rumpe, B., Schuerr, A., Tratt, L.: Model transformations in practice workshop. In Bruel, J.M., ed.: Satellite Events at the MoDELS 2005 Conference. Volume 3844. Springer (January 2006) 120–127
10. Hermann, F., Ehrig, H., Golas, U., Orejas, F.: Efficient Analysis and Execution of Correct and Complete Model Transformations Based on Triple Graph Grammars. In Bézivin, J., Soley, R., Vallecillo, A., eds.: MDI'10, ACM (2010) 22–31
11. Ehrig, H., Ermel, C., Hermann, F.: Transformation of Type Graphs with Inheritance for Ensuring Security in E-Government Networks. In Wirsing, M., Chechik, M., eds.: Proc. International Conference on Fundamental Aspects of Software Engineering (FASE'09). Volume 5503 of LNCS., Springer (2009) 325–339
12. Nachtigall, N., Hermann, F., Braatz, B., Engel, T.: Towards Domain Completeness for Model Transformations Based on Triple Graph Grammars - Extended Version. Technical Report TR-SnT-2014-14, University of Luxembourg, SnT (2014)
13. TFS-Group, TU Berlin: AGG. (2014) http://www.tfs.tu-berlin.de/agg.
14. Hermann, F., Ehrig, H., Orejas, F., Czarnecki, K., Diskin, Z., Xiong, Y.: Correctness of model synchronization based on triple graph grammars-extended version. Technical Report 2011-07, TU Berlin, Fak. IV (2011)
15. Giese, H., Hildebrandt, S., Lambers, L.: Bridging the gap between formal semantics and implementation of triple graph grammars. Software & Systems Modeling **13**(1) (2014) 273–299
16. de Lara, J., Guerra, E.: Pattern-Based Model-to-Model Transformation. In Ehrig, H., Heckel, R., Rozenberg, G., Taentzer, G., eds.: Proc. 4th Int. Conf. on Graph Transformations (ICGT 2008). Volume 5214 of LNCS., Springer (2008) 426–441
17. Orejas, F., Guerra, E., de Lara, J., Ehrig, H.: Correctness, Completeness and Termination of Pattern-Based Model-to-Model Transformation. In Kurz, A., Lenisa, M., Tarlecki, A., eds.: Int. Conf. on Algebra and Coalgebra in Computer Science (CALCO'09). Volume 5728 of LNCS., Springer (2009) 383–397
18. Cabot, J., Clarisó, R., Guerra, E., de Lara, J.: Verification and validation of declarative model-to-model transformations through invariants. J. Syst. Softw. **83**(2) (February 2010) 283–302