A generalization of termination conditions for partial model completion

Fazle Rabbi[ab], Yngve Lamo[a], Lars Michael Kristensen[a], and Ingrid Chieh Yu[b]

[a] Bergen University College, Bergen, Norway
[b] University Of Oslo, Oslo, Norway
fra@hib.no, Yngve.Lamo@hib.no, lmkr@hib.no, ingridcy@ifi.uio.no

1 Introduction

It is ironic that Model Driven Engineering (MDE) was introduced to reduce the complexity of system development, but in many cases, adds accidental complexity [7]. In the process of development, software designers are often confronted with a variety of inconsistencies and/or incompleteness in the models under construction [4]. In particular, the modeller will most of the time be working with a *partial model* not conforming (i.e., being typed by and satisfying modelling constraints) to the metamodel that defines the modelling language being used [6]. Clearly, the productivity of the modeller could be improved by providing editing support that could either automatically fix a partial model or make suggestions based on *completion rules* to assist the modeller in completing the model [5]. In many respects, this idea is similar to code completion features as found in IDEs. More generally, complexity of modelling could be reduced by providing editing support for automated rewriting of models so that they conform to the modelling language used. However the philosophy of this approach incorporates an important element: in the form of 'termination analysis'. In order to guarantee termination of the model completion, we propose a set of sufficient termination criteria.

1.1 Example of a model completion

In [5] we presented a web-based metamodelling and transformation tool called WebDPF based on the Diagram Predicate Framework (DPF) [2] which supports multilevel metamodelling and allows partial model completion. DPF provides an abstract visualization of concrete constraints. In WebDPF, one can graphically specifiy completion rules. WebDPF exploits the locality of model transformation rules and provides a foundation that enable automated tool-support to increase modelling productivity. The WebDPF editor (see Figure 1) consists of four resizable windows. The windows are arranged in a single view which provides the modeller with an overview of different modelling artefacts. The control panel on the left allows the user to select metamodels from the metamodel stack and also provides options to perform analysis such as conformance checking and termination. The conformance checking is used for validating whether a model conforms to its metamodel and the termination analysis is used for checking whether the application of transformation rules are terminating. While designing a model using the WebDPF model editor, the metamodel viewer displays the metamodel to help the modeller choose types for modelling elements. The signature editor is used to graphically define the arity and visualization of predicates. An atomic constraint can be defined by selecting a predicate in the signature editor and binding the arity of the predicate to the model elements from the model editor. Figure 1 shows a predicate called *composition* ([*comp*]) in the signature editor, and its associated completion rule in the completion rule editor. The purpose of the completion rule is to fix a model with missing edges. A partial model instance is shown in Figure 2 which

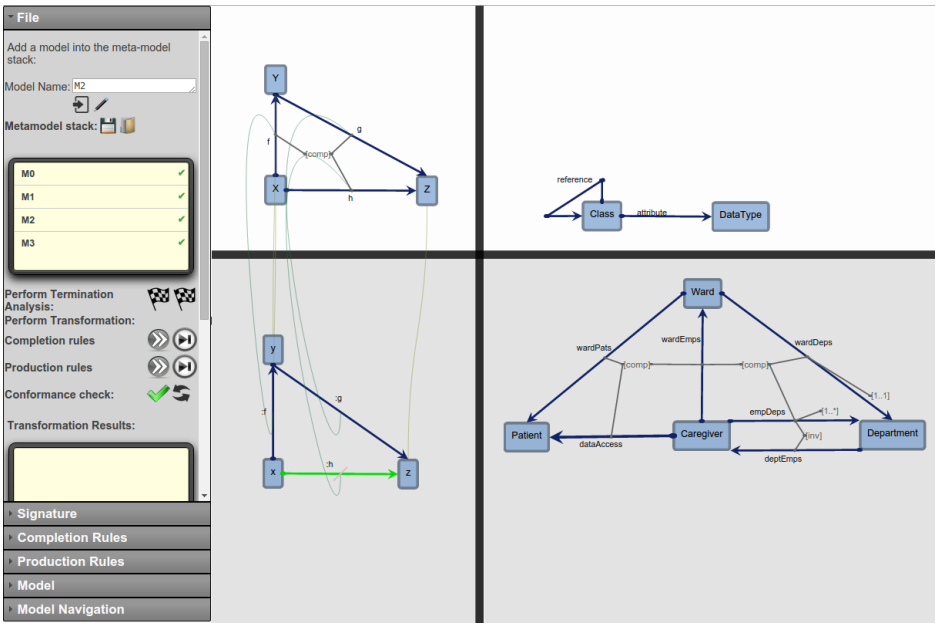

Figure 1: The WebDPF editor with a control panel (left), a metamodel viewer (top right), a model editor (bottom right), a signature editor (top middle), and a completion rule editor (bottom middle)

does not satisfy all the atomic constraints. After applying completion rules, the partial model instance becomes a complete model instance.

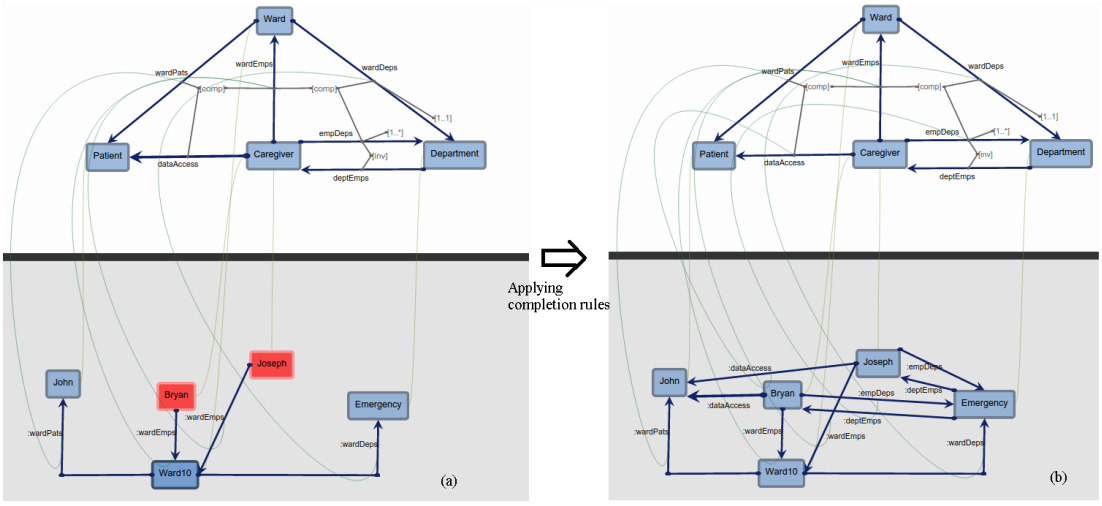

Figure 2: (a) An inconsistent model instance and (b) a complete model instance

2 Termination Criterion

In this article we focus on termination analysis for the application of transformation rules for model completion. Our proposed analysis is based on the principles adapted from layered graph grammars [1]. In a layered typed graph grammar, transformation rules are distributed across different layers. The transformation rules of a layer are applied as long as possible before going to the next layer. Ehrig et al [1] distinguished between deletion and nondeletion layers where all transformation rules in deletion layers delete at least one element and all transformation rules in nondeletion layers do not delete any elements, but the rules have negative application conditions. A set of layer conditions was specified in [1] that need to be satisfied by each layer k to guarantee termination. The layer conditions in [1] do not permit a rule r to use an element x of a given type t for the match if any element of type t has been created in a layer $k' \leq k$. The layer conditions also imply that the creation layer of an element of type t must precede its deletion layer. This restriction prevents the repetitive application of a certain rule. This layered typed graph grammar approach is suitable for situations where repetitive application of rules are not required. Unfortunately, there are many situations where repetitive application of rules are desirable such as to compute transitive closure operations [3].

To overcome the limitations of [1], discussed above, we generalize the layer conditions from [1] allowing deleting and non-deleting rules to reside in the same layer as long as the rules are loop-free. Furthermore, our generalization permits a rule to use newly created edges allowing us to perform transitive closure operations. A loop detection algorithm is implemented that overestimates the existence of a loop from a set of rules. Let R_k be the set of rules of a layer k. Our loop detection algorithm is based on the following sufficient conditions for loop freeness.

- *Cond1:* If a rule $r_i \in R_k$ creates an element x of type t, then r_i must have x in its negative application condition,
- *Cond2:* If a rule $r_i \in R_k$ deletes an element of type t, then there is no rule in $r_j \in R_k$ that creates an element of type t,

Note that we are assuming, that there are a finite number of rules in each layer and that the rules are applied on a finite input graph. The algorithm we developed guarantees termination if all the rules for each layer satisfies the above mentioned conditions. We wish to cover the following topics during our presentation:

- Generalized termination conditions
- Proof of correctness of our algorithm
- Discussion on complexity of the algorithm

References

[1] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation.* Monographs in Theoretical Computer Science. Springer, 2006.

[2] Y. Lamo, X. Wang, F. Mantz, W. MacCaull, and A. Rutle. Dpf workbench: A diagrammatic multi-layer domain specific (meta-)modelling environment. In R. Lee, editor, *Computer and Information Science 2012*, volume 429 of *Studies in Computational Intelligence*, pages 37–52. Springer, 2012.

[3] T. Levendovszky, U. Prange, and H. Ehrig. Termination criteria for dpo transformations with injective matches. *Electr. Notes Theor. Comput. Sci.*, 175(4):87–100, 2007.

[4] T. Mens and R. Van Der Straeten. Incremental resolution of model inconsistencies. In *Recent Trends in Algebraic Development Techniques*, volume 4409 of *LNCS*, pages 111–126. Springer, 2007.

[5] F. Rabbi, Y. Lamo, I. C. Yu, and L. M. Kristensen. A diagrammatic approach to model completion. In *4th Workshop on the Analysis of Model Transformations (AMT) @ MODELS'15, accepted*, 2015.

[6] S. Sen, B. Baudry, and D. Precup. Partial model completion in model driven engineering using constraint logic programming. In *INAP'07*, Germany, 2007.

[7] J. Whittle, J. Hutchinson, M. Rouncefield, H. Burden, and R. Heldal. Industrial adoption of model-driven engineering: Are the tools really the problem? In *16th International Conference, MODELS 2013*, volume 8107 of *LNCS*, pages 1–17. Springer, 2013.