# Analyzing Conflicts and Dependencies of Rule-Based Transformations in Henshin*

Kristopher Born, Thorsten Arendt, Florian Heß, and Gabriele Taentzer

Philipps-Universität Marburg, Germany
{born,arendt,hessflorian,taentzer}@informatik.uni-marburg.de

**Abstract.** Rule-based model transformation approaches show two kinds of non-determinism: (1) Several rules may be applicable to the same model and (2) a rule may be applicable at several different matches. If two rule applications to the same model exist, they may be in conflict, i.e., one application may disable the other one. Furthermore, rule applications may enable others leading to dependencies. The critical pair analysis (CPA) can report all potential conflicts and dependencies of rule applications that may occur during model transformation processes. This paper presents the CPA integrated in Henshin, a model transformation environment based on the Eclipse Modeling Framework (EMF).

## 1 Introduction

Rule-based model transformation systems can control the application of rules not only by explicit control mechanisms but also by causal dependencies of rule applications. Hence, these causal dependencies influence their execution order. If, e.g., a rule creates a model element, it can be used in subsequent rule applications. It can also happen that two rule applications overlap in a model element and one rule is to delete it while the other one requires its existence. For a better understanding of this implicit control flow, it is interesting to analyze all potential causal dependencies of rule applications for a given rule set.

The *critical pair analysis* (CPA) for graph rewriting [6] can be adapted to rule-based model transformation, e.g., to find conflicting functional requirements for software systems [7], or to analyze potential causal dependencies between model refactorings [9] which helps to make informed decisions on the most suitable refactoring to apply next. The CPA reports two different forms of potential causal dependencies, called conflicts and dependencies.
The application of a rule $r_1$ is in *conflict* with the application of a rule $r_2$ if

- $r_1$ deletes a model element used by the application of $r_2$ **(delete/use)**, or
- $r_1$ produces a model element that $r_2$ forbids **(produce/forbid)**, or
- $r_1$ changes an attribute value used by $r_2$ **(change/use)**.[1]

* This work was partially funded by the German Research Foundation, Priority Program SPP 1593 "Design for Future – Managed Software Evolution".

[1] Dependencies between rule applications can be characterized analogously.


A. Egyed and I. Schaefer (Eds.): FASE 2015, LNCS 9033, pp. 165–168, 2015.
DOI: 10.1007/978-3-662-46675-9_11

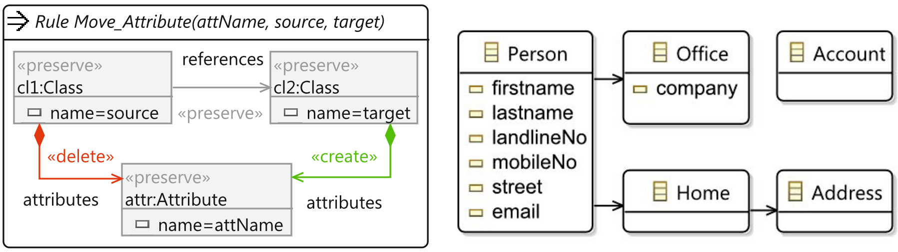
In our work, we extended Henshin [2], a rule-based model transformation language adapting graph transformation concepts and being based on the Eclipse Modeling Framework (EMF) [5]. Our extension computes all potential conflicts and dependencies of a set of rules and reports them in form of critical pairs. Each critical pair consists of the respective pair of rules, the kind of potential conflict or dependency found, and a minimal instance model illustrating the conflict or dependency. The analysis can be fine-tuned by a number of additional options to be set. The adoption of graph transformation theory to EMF model transformation requires to check the transformation rules for preserving model consistency and the resulting minimal model for being a valid EMF model [4].

The next section introduces a running example and discusses expected results; afterwards the new analysis tool is presented.

## 2 Model Transformation with Henshin

EMF is a common and widely-used open source technology in model-based software development. It extends Eclipse by modeling facilities and allows for defining (meta-)models and modeling languages by means of structured data models.

Henshin is an EMF model transformation engine based on graph transformation concepts. Since refactoring is a specific kind of model transformation, refactorings of EMF-based models can be specified in Henshin and then integrated into a refactoring framework such as EMF Refactor [3]. To demonstrate the main idea, we limit ourselves to one rule of a refactoring example for class modeling [8]. Rule `Move_Attribute` (Figure 1(a)) specifies the shift of an attribute from its owning class to an associated one along a reference. It is shown in abstract syntax. Objects and references tagged by ⟨⟨preserve⟩⟩ represent unchanged model elements, elements tagged by ⟨⟨create⟩⟩ represent new ones whereas those tagged by ⟨⟨delete⟩⟩ are removed by the transformation.




(a) (b)

**Fig. 1.** Henshin refactoring rule (a) and class model `Address Book`(b)

Modifying the class model in Figure 1(b) by the refactoring specified in Figure 1(a), we observe two potential problems: (1) The attribute `landlineNo` of class `Person` can be shifted to either class `Home` or class `Office` (by refactoring `Move_Attribute`). However, if it is shifted to class `Home` the other refactoring becomes inapplicable (and vice versa). This means, refactoring `Move_Attribute` is in conflict with itself. (2) The attribute `street` of class `Person` can be shifted to class `Address` via class `Home` (by two applications of `Move_Attribute` along existing references). The second shift is currently not possible since class `Home` does not have an attribute so far, i.e., refactoring `Move_Attribute` may depend on itself. Graph transformation theory allows us to analyze such conflicts and dependencies at specification time by relying on the idea of the CPA.

## 3 Tooling

The provided CPA extension of Henshin can be used in two different ways: Its application programming interface (API) can be used to integrate the CPA into other tools and a user interface (UI) is provided supporting domain experts in developing rules by using the CPA interactively.

After invoking the analysis, the rule set and the kind of critical pairs to be analyzed have to be specified. Furthermore, options can be customized to stop the calculation after finding a first critical pair, to ignore critical pairs of the same rules, etc. The resulting list of critical pairs is shown and ordered along rule pairs. Figure 2 depicts an example for the analysis of rule `Move_Attribute`, in which the delete/use-conflict (1) corresponds to the example discussed above.

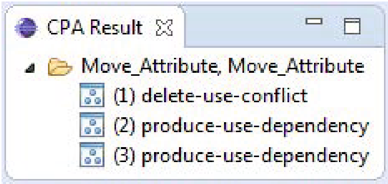



**Fig. 2.** The result view

The subsequent dependency results differ in their target of the second attribute movement. The first produce/use-dependency (2) represents the case of moving the attribute back to the original class, which leads to a smaller minimal model with only two classes referencing each other, as depicted in Figure 3. The highlighting by enclosing hash marks is the most important information, since the enclosing element is the cause of the dependency. The link between `2:Class` and `3:Attribute` is created by the first rule application and is required by the second application. Since all elements and values in the minimal model may be matched by the first and the second rule application, there is a generic approach to represent attribute values. Value `r1_source_r2_target`, e.g., means that it must conform to value `source` in rule `r1` and value `target` in rule `r2`, respectively (compare Fig. 1(a)). The second

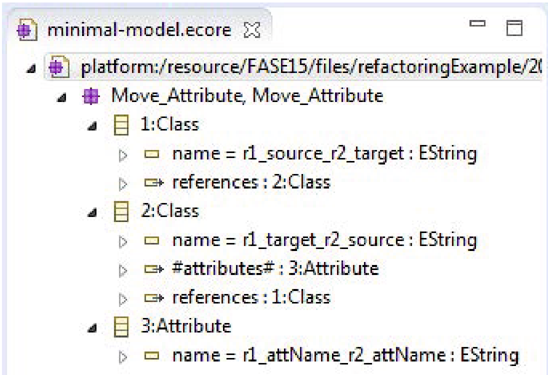



**Fig. 3.** Minimal model of a dependency

dependency reported in Figure 2 is the handling of two consecutive attribute shifts, also described in Section 2.

The current version of the tool can analyze rules with negative application conditions and attributes of primitive data types. Positive application conditions shall be supported in the future. In order to avoid improper results, the rules are checked regarding these prerequisites. Further checks ensure that the rules are consistent to the properties defined in [4]. The LHS, RHS and intersection graphs of each rule are checked to comply to Definition 3 in [4], e.g., each node must have at most one container, there is no containment cycle. Furthermore, rules have to ensure consistent results, i.e., have to comply to Def. 6 in [4], ensuring e.g. that containment edge deletion and creation is restricted to edge redirection. The rule shown in Figure 1(a) is consistent to this definition. Internally, the CPA extension of Henshin is based on the graph transformation tool AGG [1]. Dedicated exporter and importer translate EMF meta-models and Henshin rules to AGG and CPA results back to EMF models.

## 4 Conclusion

The model transformation tool Henshin has been extended by a critical pair analysis to inspect rule sets for dependencies and conflicts. An interactive user interface is provided allowing the inspection of each critical pair in detail.

For the future, we intend to support also a confluence check of critical pairs, for which the CPA is a first step. The tool download as well as additional information on the CPA in Henshin, especially with respect to the translation between Henshin and AGG, can be found at [8].

## References

1. AGG, http://user.cs.tu-berlin.de/~gragra/agg/
2. 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.) MODELS 2010, Part I. LNCS, vol. 6394, pp. 121–135. Springer, Heidelberg (2010), http://www.eclipse.org/henshin/
3. Arendt, T., Taentzer, G.: A tool environment for quality assurance based on the Eclipse Modeling Framework. Automated Software Engineering 20(2), 141–184 (2013), http://www.eclipse.org/emf-refactor
4. Biermann, E., Ermel, C., Taentzer, G.: Formal foundation of consistent EMF model transformations by algebraic graph transformation. SoSyM 11(2), 227–250 (2012)
5. Eclipse: Eclipse Modeling Framework (EMF), http://www.eclipse.org/emf
6. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Monographs in Theoretical Computer Science. Springer (2006)
7. Hausmann, J.H., Heckel, R., Taentzer, G.: Detection of Conflicting Functional Requirements in a Use Case-Driven Approach: A Static Analysis Technique Based on Graph Transformation. In: ICSE, pp. 105–115. ACM (2002)
8. Tool download and installation, http://www.uni-marburg.de/fb12/swt/cpa
9. Mens, T., Taentzer, G., Runge, O.: Analysing refactoring dependencies using graph transformation. Software and System Modeling 6(3), 269–285 (2007)