Abstract. Refinement is a central notion in computer science, meaning that some artefact $S$ can be safely replaced by a refinement $R$, which preserves $S$’s properties. Having available techniques and tools to check transformation refinement would enable (a) the reasoning on whether a transformation correctly implements some requirements, (b) whether a transformation implementation can be safely replaced by another one (e.g. when migrating from QVT-R to ATL), and (c) bring techniques from stepwise refinement for the engineering of model transformations.

In this paper, we propose an automated methodology and tool support to check transformation refinement. Our procedure admits heterogeneous specification (e.g. PAMoMO, Tracts, OCL) and implementation languages (e.g. ATL, QVT), relying on their translation to OCL as a common representation formalism and on the use of model finding tools.

1 Introduction

The raising complexity of languages, models and their associated transformations makes evident the need for engineering methods to develop model transformations [12]. Model transformations are software artefacts and, as such, should be developed using sound engineering principles. However, in current practice, transformations are normally directly encoded in some transformation language, with no explicit account for their requirements. These are of utmost importance, as they express what the transformation has to do, and can be used as a basis to assert correctness of transformation implementations. While many proposals for requirements gathering, representation and reasoning techniques have been proposed for general software engineering [15,23], their use is still the exception when developing model transformations.

Specifications play an important role in software engineering, and can be used in the development of model transformations in several ways. First, they make explicit what the transformation should do, and can be used as a basis for implementation. Specifications do not necessarily need to be complete, but can document the main requirements and properties expected of a transformation. Then, they can be used as oracle functions for testing implementations [11].
In this setting, it is useful to know when a transformation $T$ refines a specification $S$. Intuitively, this means that $T$ can be used in place of $S$ without breaking any assumption of the users of $S$. Some other times, we need to know whether a transformation $T$ refines another transformation $T'$ and can replace it. Fig. 1 gathers several scenarios where checking transformation refinement is useful. In (a), an implementation refines a requirements specification, hence ensuring correctness of the implementation with respect to the specification. In (b), a transformation implementation (e.g. in QVT) is refined by another one (e.g. in ATL) which can replace the former safely. This is especially useful if we want to migrate transformations, ensuring correctness of the migrated transformation. Finally, in (c), a specification refines another specification, which enables the application of stepwise refinement methodologies for transformation development.

In this paper, we tackle the previous scenarios by proposing an automated methodology to check transformation refinements. Our proposal relies on OCL as a common denominator for both specification languages (e.g. PAMoMo [13], Tracts [25] and OCL [18]) and transformation languages (e.g. QVT-R [21], triple graph grammars [22] and ATL [16]). For this purpose, we profit from previous works translating these languages into OCL [6,7,13,25]. Hence, transformation specifications and implementations are transformed into transformation models [4] and we use SAT/model finding [6] techniques to automatically find counterexamples that satisfy properties assumed by the specification, but are incorrectly implemented. While refinement has been previously tackled in [25], our work is novel in that it proposes an automated procedure for performing this checking, and is able to tackle heterogeneous specification and transformation languages by using OCL as the underlying language for reasoning.

Paper organization. Section 2 motivates the need for transformation refinement using an example. Section 3 introduces model transformation refinement. Section 4 details our methodology to check refinements. Section 5 provides more examples, Section 6 compares with related work and Section 7 concludes.

2 A Motivating Example

Assume we have gathered the requirements for the Class2Relational transformation, and want to use them as a blueprint to check whether an implementation correctly addresses them. Fig. 2 shows part of a specification of the requirements using the PAMoMo specification language [13], though we could choose any other transformation specification language instead (like Tracts or OCL).

PAMoMo is a formal, pattern-based, declarative, bidirectional specification language that can be used to describe correctness requirements of transformations and of their input and output models in an implementation-independent...