Traceability Mappings as a Fundamental Instrument in Model Transformations

Zinovy Diskin[1], Abel Gómez[2], and Jordi Cabot[2,3]

[1] McMaster University, Canada
diskinz@mcmaster.ca
[2] IN3, Universitat Oberta de Catalunya, Spain
agomezlla@uoc.edu
[3] ICREA, Spain
jordi.cabot@icrea.cat

Abstract. Technological importance of traceability mappings for model transformations is well-known, but they have often been considered as an auxiliary element generated during the transformation execution and providing accessory information. This paper argues that traceability mappings should instead be regarded as a core aspect of the transformation definition, and a key instrument in the transformation management. We will show how a transformation can be represented as the result of execution of a metamodel mapping, which acts as a special encoding of the transformation definition. Since mappings enjoy Boolean operations (as *sets* of links) and sequential composition (as sets of *directed* links), encoding transformations by mappings makes it possible to define these operations for transformations as well, which can be useful for model transformation reuse, compositional design, and chaining.

1 Introduction

Translating models from one to another metamodel (also known as model-to-model transformation, MMT) is ubiquitous in software engineering. The technological importance of traceability for MMT is well recognized in the MMT community. Such widely used transformation languages as ATL [11,16] and ETL [12,17] automatically create traceability links during the transformation execution in order to resolve dependencies between the rules, and perhaps for debugging and maintenance. Moreover, a traceability mapping (i.e., a set of links) between the metamodels can be used as an MMT definition, which may be immediately executed [9,13,14], or used for automatic transformation code generation [8].

Here, we present a theoretical framework, in which traceability aspects of MMT are precisely discussed: we specify execution of metatraceability mappings as an abstract mathematical operation, show the importance of several concepts, which are underestimated or missing from the literature, and derive some practical recommendations on MMT management, including transformation chaining.

In Sect. 2, we show that semantics of MMTs without traceability mappings is essentially incomplete: we present an example of two different transformations

indistinguishable in the traceability-free setting. We then demonstrate that traceability links between the source and the target models work in concert with the respective links between the source and the target metamodels, which means commutativity of the respective diagrams. Moreover, execution of a metamodel mapping for a given source model can be specified as an algebraic operation (called *pullback* in category theory), for which commutativity is central.

However, simple traceability mappings considered in Sect. 2, which relate two metamodels with similar structures, do not cover many practically interesting cases of structurally different metamodels. To address the problem of executing metamodel mappings relating metamodels with different structures, several complex approaches have been developed [9,19]. We propose a simpler solution, in which we first augment the source metamodel with derived elements that make its structure similar to the target metamodel, and then relate the two by a simple mapping. The derived elements actually encode operations/queries against the source metamodel, which can be executed for any model instantiating it. Hence, a complex metamodel mapping is executed in two steps: first the query is executed, then the mapping as such is executed. The approach is discussed in Sect. 3, and in Sect. 5 we show how it can be implemented with ATL.

Thus, even complex MMTs are encoded in a unified and transparent way as mappings relating metamodels with, perhaps, derived elements/queries involved. An important consequence of this encoding is that we can employ well studied algebraic operations over mappings for manipulating MMTs as black-boxed objects. Specifically, being *sets* of links, mappings enjoy intersections and union operations over them, which can be employed for reuse. Being sets of *directed* links, mappings can be sequentially composed, and the respective transformations chained (which is considered to be a challenging task). Particularly, chaining can be employed for incremental compositional design of MMTs. Algebra of MMTs is considered in Sect. 4, and in Sect. 5 we show how it can be applied for manipulating ATL transformations. Finally, we observe related work in Sect. 6, and conclude in Sect. 7.

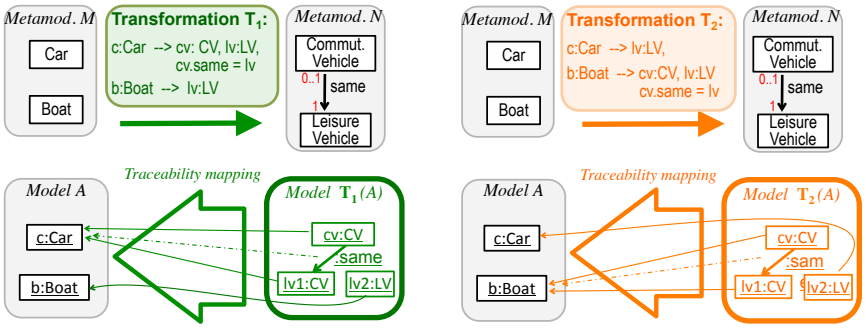
2 Analysing traceability mappings

We show the semantic necessity of traceability mappings in Sect. 2.1, consider their properties and representation in Sect. 2.2, and execution in Sect. 2.3.

2.1 The semantic necessity of traceability mappings

Semantics of a model-to-model transformation $\mathbf{T}$ is commonly considered to be a function $[\![\mathbf{T}]\!]\colon [\![M]\!] \to [\![N]\!]$, where $[\![M]\!]$ and $[\![N]\!]$ are model spaces defined by, resp., the source, M, and the target, N, metamodels. However, in this section we present two different transformations generating the same model space mapping, and then call traceability mappings to the rescue.

Figure 1(a) presents a toy transformation example. The source metamodel M specifies two classes, Car and Boat, and the target metamodel N specifies their



(a) (b)

Fig. 1: Two sample transformations.

possible roles as Commuting and Leisure Vehicles, connected by association same if two roles are played by the same physical vehicle; e.g., such a transformation may be needed for an insurance company. The transformation $\mathbf{T}_1$ consists of two rules specified in Fig. 1(a) in some pseudo MT language. The first rule says that a car produces a commuting vehicle and a leisure vehicle connected by a same-link. The second rule says that a boat generates a leisure vehicle. An example of executing this transformation for a model A consisting of a car and a boat is shown in the lower half of Fig. 1(a), where $\mathbf{T}_1(A)$ denotes the target model produced by the transformation; ignore the mapping from $\mathbf{T}_1(A)$ to A for the moment. Here we write $\mathbf{T}(A)$ for $[\![\mathbf{T}]\!](A)$. We will often abuse such a notation and use the same symbol for a syntactic construct and its intended semantics.

Figure 1(b) presents a different transformation $\mathbf{T}_2$. Now a boat gives rise to a commuting and a leisure vehicle, whereas a car only produces a leisure vehicle (think about people living on an island). Clearly, being executed for the same source model A, transformation $\mathbf{T}_2$ produces the same target model consisting of three objects and a same-link. More accurately, models $\mathbf{T}_1(A)$ and $\mathbf{T}_2(A)$ are isomorphic rather than equal, but the same transformation $\mathbf{T}$ executed twice for the same model A would also produce isomorphic rather than equal models as MMTs are normally defined up to OIDs. We will always understand equality of models up to OID isomorphism, and thus can write $\mathbf{T}_1(A) = \mathbf{T}_2(A)$. It is easy to see that such an equality will hold for any source model containing equal numbers of cars and boats. If wesuppose that the metamodel M includes a constraint requiring the numbers of cars and boats to be equal (eg, has an association between classes Car and Boat with multiplicity 1..1 at both ends), then any instance X of M would necessarily consist of equal numbers of cars and boats. Hence, $\mathbf{T}_1(X) = \mathbf{T}_2(X)$ holds for *any* source instance $X \in [\![M]\!]$.

Thus, the common semantics of a model transformation as a model space mapping $[\![\mathbf{T}]\!]: [\![M]\!] \to [\![N]\!]$ is too poor and should be enriched. Comparison of the two transformations in Fig. 1(a,b), now with traceability mappings, shows what should be done: we need to include traceability mappings into the semantics of MMTs, and define it as a function $[\![\mathbf{T}]\!]: [\![M]\!] \to [\![N]\!] \times Map([\![N]\!],[\![M]\!])$, where $Map([\![N]\!],[\![M]\!])$ denotes the set of all mappings from N-models (elements of $[\![N]\!]$)

to M-models (in $[\![M]\!]$). It is convenient to split semantics into two functions:

$$[\![\mathbf{T}]\!]^{\bullet}\colon\ [\![M]\!] \to [\![N]\!] \text{ and } [\![\mathbf{T}]\!]^{\blacktriangleleft}\colon\ [\![M]\!] \to Map([\![N]\!],[\![M]\!])$$

such that for any source model A, mapping $[\![\mathbf{T}]\!]^{\blacktriangleleft}(A)$ is directed from model $[\![\mathbf{T}]\!]^{\bullet}(A)$ to A (as suggested by the tringle superindex). Thus, what is missing in the common MMT-semantics is the mapping-valued function $[\![\mathbf{T}]\!]^{\blacktriangleleft}$. Including this function into semantics has several important consequences discussed below. Below we will use a simplified notation with semantic double-brackets omitted.

2.2 Traceability under the microscope

We discuss properties of traceability mappings: structure preservation, commuting with typing, and their span representation. As an example, we use transformation $\mathbf{T}_1$ from Fig. 1(a), but denote it by $\mathbf{T}$ to avoid excessive indexing.

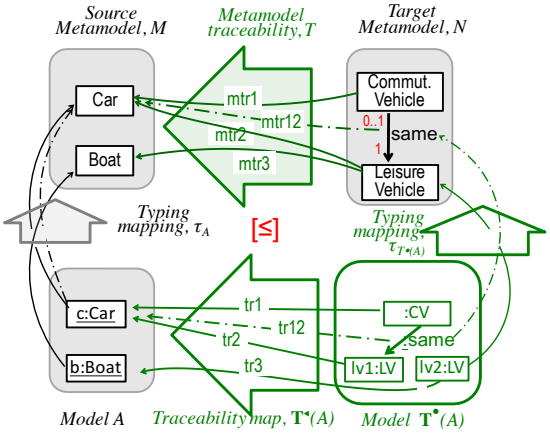


Fig. 2: Meta-traceability

2.2.1 Structure preservation. A mapping is a collection of directed links that is compatible with models' structure. If models are graphs, then their graph structure, i.e., the incidence of nodes and edges, should be respected. Consider, e.g., the lower mapping $\mathbf{T}^{\blacktriangleleft}(A)$ in Fig. 2, which reproduces the respective traceability mapping in Fig. 1(a). The dashed link from edge same to node c:Car in this mapping actually denotes a link targeted at the *identity* loop of the node c, which relates c to itself. Such loops can be added to every graph node, and when we draw a link from an arrow to a node, it is just a syntactic sugar to specify a link targeted at the node's identity loop. With this reservation, it is seen that both traceability mappings in Fig. 1 are correct graph morphisms, which map nodes to nodes and edges to edges so that the incidence between nodes and edges is preserved.

2.2.2 Meta-traceability and commutativity. Another important condition to be respected is compatibility of links between model elements with relationships between metamodel elements established by the transformation definition. To explicate the latter, we need *meta-traceability* links between metamodels as shown in the upper half of Fig. 2. The mapping T consists of four links mtr_i $(i = 1, 12, 2, 3)$ "tracing" the origin of the target metamodel elements according

to the (green) transformation definition in Fig. 1(a): commuting vehicles appear from cars (rule 1) and only from cars (neither of the other rules produce commuting vehicles), and leisure vehicles appear either from cars (rule 1) or boats (rule 2). The dashed link to Car again denotes a formal link from edge same to not-shown identity loop link from Car to Car, and encodes the clause in rule 1 that a same-link appears when a car generates both a commuting and leisure vehicle. Thus, the upper three meta-links in mapping T "trace" rule 1 in transformation $\mathbf{T}$, and the lower link "traces" rule 2.

Now recall that a model A is actually a pair (D_A, τ_A) with D_A the model's datagraph, and τ_A: $D_A \to M$ a *typing* mapping. By a common abuse of notation, we denote the datagraph by the same letter A as the entire model. In Fig. 2, two typing mappings (two vertical block-arrows) and two traceability mappings (two horizontal block-arrows) form a square, and this square is *semi-commutative* in the sense that the following two conditions hold. First, each of the four paths from $\mathbf{T}^{\bullet}(A)$ to A (via traceability links tr_i) to M (via A's type links) can be matched by a same-source-same-target path from $\mathbf{T}^{\bullet}(A)$ to N (via type links) to M (via meta-traceability links mtr_i). Second, there is an upper path without match, namely, the path from object $lv1 \in \mathbf{T}^{\bullet}(A)$ to class LeisureVehicle to class Boat (hence the $\leq$ symbol denoting this property of the square diagram). As commutativity rather than semi-commutativity is an important ingredient of the mapping machinery, we need to fix the commutativity violation. The next sections shows how to do it.

2.2.3 Traceability mappings via spans. As traceability links are fundamental, we reify them as model elements, and collect these elements in model $|T|$ in the upper part of Fig. 3. Three nodes in this model reify links $mtr_{1,2,3}$ between nodes in the metamodels (see Fig. 2), and the arrow in $|T|$ reifies the (dashed) link tr_{12} between arrows in the metamodels (recall that the actual target of this link is the identity loop of the targte node). In this way we build a metamodel $|T|$ consisting of three classes and one association. The special nature of $|T|$'s elements (which are, in fact, links) is encoded by mapping each element to its ends in metamodels M and N. These secondary links form totally-defined single-valued mappings T_M: $M \leftarrow |T|$ and T_N: $|T| \to N$ so that we replaced a many-to-many mapping T by a pair of single-valued (many-to-one) mappings. Working with single-valued mappings is usually much simpler technically, and below we will see that it allows us to fix commutativity.

The triple $T = (T_M, |T|, T_N)$ is called a *span* with the *head* $|T|$, and *legs* T_M and T_N. We will call the first leg in the triple the *source* leg, and the second one the *target* leg. Thus, span T in Fig. 3 encodes mapping T in Fig. 2 (and they are thus denoted by the same letter). We will also use the same letter for the head of the span to reduce the number of symbols in our formulas. Note that the head of the span is a graph (because mapping T is a graph mapping), and its legs are correct graph morphisms. This is an accurate formalization of the structure preservation property discussed in Sect. 2.2.1.

The reification procedure applied to mapping $\mathbf{T}^{\blacktriangleleft}(A)$ (Fig. 2) provides the span shown in the lower part of Fig. 3. We denote its head by $T^{\bullet}_M(A)$ rather than

by $\mathbf{T}^{\blacktriangleleft}(A)$, as in the next subsection we will show how this model (and mapping $\mathbf{T}^{\blacktriangleleft}_M$) can be computed from the left half of the upper span and model A, and by the same reason, these elements are blank (and blue with a color display) rather than shaded (and black)—ignore these details for a moment. We also omitted all vertical links constituting typong mappings (shown by vertical block arrows).

Since in contrast to mapping T, mapping $\mathbf{T}^{\blacktriangleleft}(A)$ in Fig. 2 is many-to-one, the right leg of the span is an isomorphism (of graphs), which we show as a block-rectangle rather than a block-arrow (actually we could identify the two models). Now it is easy to check commutativity of the two square diagrams, which is recorded by markers [=] at their centers. Commuting makes it possible to type elements in model $|\mathbf{T}^{\blacktriangleleft}(A)|$ (i.e., traceability links) by elements in model $|T|$ (i.e., meta-traceability links), and ensures that typing is a correct graph morphism. We have thus obtained an accurate formal specification of mutually consistent traceability mappings.

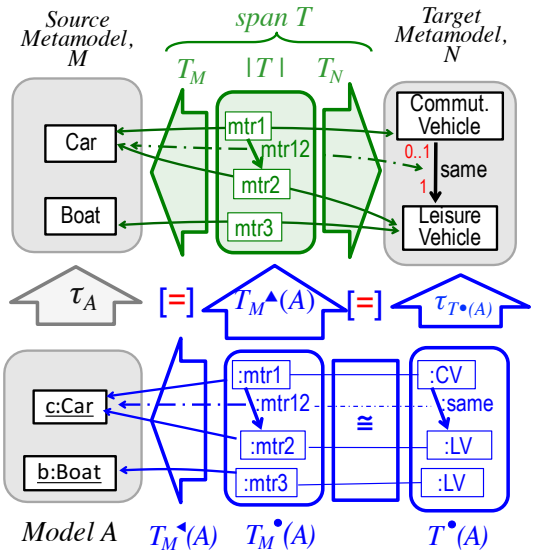


Fig. 3: Meta-traceability via spans

Span T (the upper half of Fig. 3) is presented in Fig. 4 in a MOF-like way via metamodeling. In these terms, meta-traceability links are classifiers for model traceability links, and commutativity conditions in Fig. 3 provide consistency of traceability links' classification with model elements' classification.

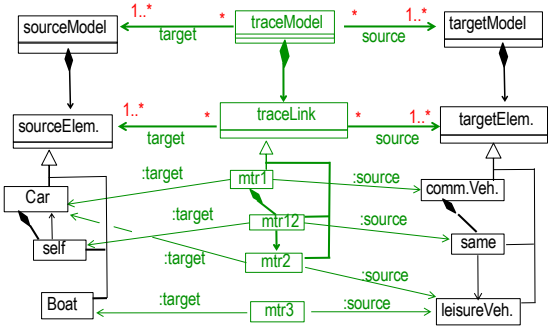


Fig. 4: Trace-links metamodel

2.3 Meta-traceability links can be executed!

A somewhat surprising observation we can make now is that the meta-traceability mapping can actually replace the transformation definition $\mathbf{T}$: by applying two

standard categorical operations to the span T and the typing mapping of model A, we can produce model $\mathbf{T}^{\bullet}(A)$ (together with its typing) and the traceability mapping $\mathbf{T}^{\blacktriangleleft}(A)$ in a fully mechanized way.

The first operation is called (in categorical jargon) *pull-back (PB)*. Its takes as its input two graph mappings with a common target, T_M and τ_A (such configuration is called a *cospan*), and outputs a span of graph mapping shown in Fig. 3 blank and blue (to recall the mechanic nature of the operation) so that the entire square diagram is commutative. The PB works as follows. For any pair of elements $a{\in}A$ and $n{\in}N$ such that there is an element $m{\in}M$ together with a pair of links (ℓ_1, ℓ_2) targeted at it, $\ell_1\colon a \to m$ in mapping τ_A and $\ell_2\colon m \leftarrow n$ in mapping T_M, an object (a, n) is created together with *projection* links to a and n. All such pairs (a, n) together with projection links to N make a model $T^{\bullet}_M(A)$ (whose typing mapping is denoted by $T^{\blacktriangle}_M(A)$ – note the tringle pointing upward), and projection links to A constitute its traceability mapping $T^{\blacktriangleleft}_M(A)$. The entire operation can be seen as ***p***ulling the model A together with its typing mapping ***b***ack along mapping T_M, hence, the name PB. Note that commutativity of the left square now becomes the very essence of the transformation: we build model $T^{\bullet}_M(A)$ and its traceability mapping in such a way that commutativity holds. Moreover, we make this model the maximal model that respect commutativity by collecting in $T^{\bullet}_M(A)$ *all* pairs (a, n) that respect commutativity. For example, if model A would have three cars and boats, model $T^{\bullet}_M(A)$ would have three commuting and five leisure vehicles with three same-links.

The second operation is fairly easy: we sequentially compose mappings $T^{\blacktriangle}_M(A)$ and T_N by composing their links, and obtain a mapping $T^{\bullet}_M(A) \to N$ that provides graph $T^{\bullet}_M(A)$ with typing over N. We will denote the model whose datagraph is $T^{\bullet}_M(A)$ (or its isomorphic copy up to OIDs) and typing map is composition $T^{\blacktriangle}_M(A); T_N$ by $T^{\bullet}(A)$. The right square in Fig. 3 illustrates this specification. It is now seen that PB followed by composition produce exactly the same model as rule-based definition $\mathbf{T}_1$ in Fig. 1(a), and the span with head $T^{\bullet}_M(A)$ is exactly the reified traceability mapping $\mathbf{T}^{\blacktriangleleft}_1(A)$ from Fig. 1(a).

3 Transformations via mappings and queries

Pulling a source model A back along a meta-traceability mapping T as described above covers a useful but not too wide class of transformations; more complex transformations need a more expressive mechanism. In [3,4], it was proposed to separate an MT into two parts: first, a complex computation over the source model is encoded by a query against the source metamodel, and then the result is relabeled (with, perhaps, multiplication) by the target metamodel according to the meta-traceability mapping.

We will illustrate how the machinery works by encoding the same transformation $\mathbf{T}$ by a different type of meta-traceability mapping employing queries against the source metamodel as shown in Fig. 5. The first basic idea of the transformation—creation of commuting vehicles by cars only—is encoded by direct linking class Commut.Vehicle to class Car as we did before. The second idea—creation of leisure vehicles by both cars and boats—is now encoded in two

steps. First, we augment the source metamodel M with a derived class Car + Boat computed by applying the operation (query) of taking the disjoint union of two classes; we denote the augmented metamodel by $Q(M)$ with Q referring to the query (or a set of queries) used for augmentation. Second, we link class LeisureVehicle to the derived class Car + Boat, and association same in metamodel N is linked to its counterpart in metamodel $Q(M)$, as shown in Fig. 5.

All links have a clear semantic meaning: given a link *qmtr* from an element n of N to an element m on $Q(M)$, we declare that n is to be instantiated exactly as m is instantiated, that is, for any model A, every element instantiating m in A or $Q(A)$ (see below), generates an element instantiating n in $T_1^\bullet(A)$. Note also that the mapping is of one-to-one type: two classes responsible for LeisureVehiclegeneration now contribute to a single query, and two respective links (*mtr2* and *mtr3* in Fig. 2) are replaced by one link *qmtr2* into the query.

Execution of the transformation for a model A also goes in two steps. First, the query used in the mapping definition is executed for the model. In our example, we take the disjoint union of Car and Boat instantiations in A, i.e., the set $\{c', b'\}$. A reasonable implementation would add a new type Car + Boat to the same object c rather than creating a new object c', but the pair (c, Car) is still different from pair $(c, \mathsf{Car} + \mathsf{Boat})$. Thus, it may happen that $c' = c$ and $b' = b$, but this is just a special case of a general pattern presented in Fig. 5. Second, objects c, c', b' are retyped according to the respective meta-traceability links $qmtr_1$ (for c) and $qmtr_2$ (for c' and b'). The link cc' is also retyped along the link *qmtr12*.

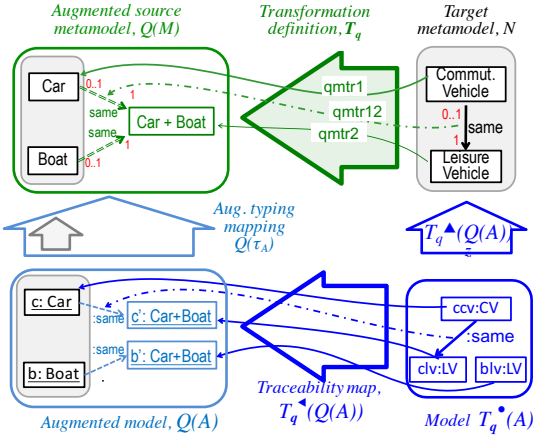


Fig. 5: Meta-traceability via queries

Thus, a model transformation definition is divided into two parts: finding a query (or a set of queries) Q against the source metamodel M, which captures the computationally non-trivial part of the transformation, and then mapping the target metamodel into the augmentation $Q(M)$, which shows how the results of the computation are to be retyped into the target metamodel. The second part can capture some simple computations like multiplication of objects (which often appears in MMT), but not more. In contrast, with a broad understanding of queries as general operations, the first part is Turing complete with the only reservation that all result of the computation must have new types (which distinguishes queries from updates).

A formal abstraction of the example is described in Fig. 6(a). A model transformation is considered to be a pair $T = (Q_T, m_T)$ with Q_T a query against the

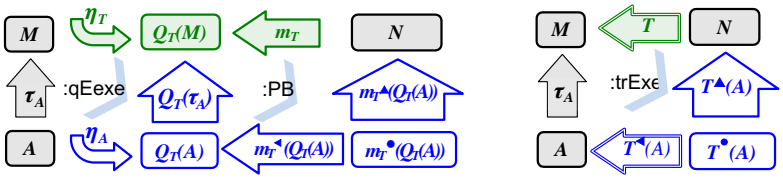


Fig. 6: Execution of meta-traceability mappings. Derived elements are blank.

source metamodel M and $m_T\colon Q_T(M) \leftarrow N$ a mapping from the target metamodel N to model M augmented with derived elements specified by the query. Formally, we have an inclusion $\eta_T\colon\ M \hookrightarrow Q_T(M)$. Note that construct Q_T is a query *definition*, which can be executed for any data conforming to schema M, i.e., for any model properly typed over the metamodel M. Execution is modeled by an operation qExe, which for a given query Q_T and model A produces an augmented model $Q_T(A)$[4] properly typed over the augmented metamodel by an augmented typing mapping $Q_T(\tau_A)$. To complete the transformation, the result of the query is retyped according to the mapping m_T (retyping is given by pulling back the augmented typing mapping as discussed above). In Fig. 6(b), an abstract view of Fig. 6(a) is presented, in which the upper double arrow encodes the sequential composition of the two upper arrows in diagram (a), and operation trExe of *transformation execution* is a composition of two operations in diagram (a). Paper [5] presents an accurate categorical formalization of this construction by modeling the query language as a *monad* and the transformation definition mapping T as a *Kleisli mapping* over this monad. We do not need formal details in this paper, but we will use the term Kleisli mapping to refer to mappings such as $m_T\colon Q_T(M) \leftarrow N$. By an abuse of notation, we will often use the same symbol T for both transformation T and its mapping m_T.

4 An Algebra for model transformations

Mappings have a dual nature. As *sets* of links, mappings are amenable to Boolean operations (union, intersection, difference). As sets of *directed* link, mappings can be sequentially composed in the associative way. Hence, representing a model transformation by a mapping allows us to build an algebra of useful operations over model transformations. For example, Boolean operations allow for reuse, and sequential composition establishes a mathematical approach to model transformation chaining. In this section, we briefly and informally consider, first, Boolean operations, and then sequential composition.

4.1 Boolean operations for model transformations

Consider our two transformations, $\mathbf{T}_1$ and $\mathbf{T}_2$, each consisting of two rules, described in Fig. 1(a,b). Although rules are related, they are all different, and,

[4] We should write $[\![Q_T]\!](A)$ but we again use the same symbol for both syntactic and semantic constructs.

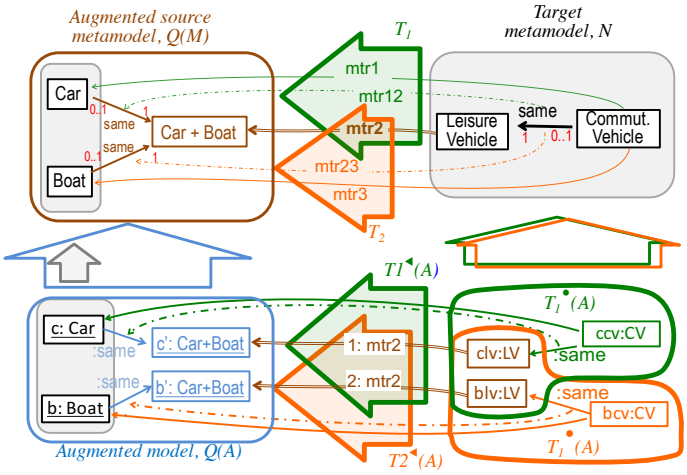


Fig. 7: BA for traceability mappings

formally speaking, for model transformation understood as sets of rules, we have $\mathbf{T}_1 \cap \mathbf{T}_2 = \varnothing$. Hence, specifying commonalities and differences between transformations needs going inside the rules and working on a finer granularity level, which may be not easy for complex rules. In contrast, encoding transformation definitions by mappings, i.e., sets of (meta) links, makes them well amenable to the variability analysis.

Mappings $T_{1,2}$ encoding transformations $\mathbf{T}_{1,2}$ resp. are shown in Fig. 7 (colored green and resp. orange with a color display). Each mapping consists of three links, and one link (double-lined and brown) is shared. This link constitutes the intersection mapping, whose domain consists of the only class LeisureVehicle (in the categorical jargon, this mapping is called the *equalizer* of T_1 and T_2). Thus, $T_{1\wedge 2} = T_1 \cap T_2 = \{mtr2\}$. We can also merge T_1 and T_2 into mapping $T_{1\vee 2} = T_1 \cup T_2$ consisting of five links. It is easy to see that for our example, execution of mappings via pullbacks is compatible with these operations so that $T^{\blacktriangleleft}_{1\wedge 2}(A) = T^{\blacktriangleleft}_1(A) \cap T^{\blacktriangleleft}_2(A)$ and $T^{\blacktriangleleft}_{1\vee 2}(A) = T^{\blacktriangleleft}_1(A) \cup T^{\blacktriangleleft}_2(A)$; particularly $T^{\bullet}_{1\wedge 2}(A) = T^{\bullet}_1(A) \cap T^{\bullet}_2(A)$ and $T^{\bullet}_{1\vee 2}(A) = T^{\bullet}_1(A) \cup T^{\bullet}_2(A)$ (note the boot-like shapes of $T^{\bullet}_i(A)$ $(i = 1, 2)$ and their intersection consisting of two objects).

The simple and appealing algebraic picture described above does not hold if transformations are seen as sets of rules as described in Fig. 1: then, as mentioned, $\mathbf{T}_1 \cap \mathbf{T}_2 = \varnothing$, and transformation $\mathbf{T}_1 \cup \mathbf{T}_2$ would translate model A into a model consisting of six rather than four objects (consider disjoint union of model $\mathbf{T}_1(A)$ and $\mathbf{T}_2(A)$ presented in Fig. 1(a,b)), which seems not well matching the intuition of how the merged transformation should work. Indeed, the two transformations differ in how commuting vehicles are generated, but agree on leisure vehicles. This agreement is exactly captured by shared link *mtr2* in mappings $T_{1,2}$, but is not taken into account in $\mathbf{T}_1 \cup \mathbf{T}_2$.

Nevertheless, suppose we still want to merge the two transformations in a disjoint way so that the merged transformation would produce a six element

model from model A. We can do it with the mapping representation as well by defining a new mapping T_{1+2} via disjoint union $T_1 + T_2$, in which link *mtr2* is repeated twice: imagine a version of Fig. 7, in which one copy of link *mtr2* belongs to mapping T_1, and the other copy belongs to T_2. It is easy to check that PB applied to mappings $Q(\tau_A)$ and T_{1+2} would result in the disjoint union of models $\mathbf{T}_1^\bullet(A)$ and $\mathbf{T}_2^\bullet(A)$ with the respective disjoint union of their traceability mappings (see Fig. 1) as required. Thus, $(T_1 + T_2)^\bullet(A) = T_1^\bullet(A) + T_1^\bullet(A)$ and $(T_1 + T_2)^\blacktriangleleft(A) = T_1^\blacktriangleleft(A) + T_1^\blacktriangleleft(A)$, and our traceability execution procedure is compatible with disjoint union as well.

4.2 Sequential composition of model transformations

Suppose that transformation $\mathbf{T}_1$ is followed by a transformation $\mathbf{T}_3$ from metamodel N to metamodel O consisting of the only class Vehicle (Fig. 8). This transformation is defined by two rules: every object of class Commut.Vehicle generates a Vehicle-object, and every LeisureVehicle-object generates a Vehicle-object too. Mapping T_3 encoding this transformation would consist of the only link mapping class Vehicle to the disjoint union of Commut.Vehicle and LeisureVehicle . To chain the transformations, we need to compose mappings T_3 and T_1 but they are not composable: mapping T_1 is not defined for the target class of mapping T_3.

The problem can be fixed if we apply the query Q_3 to the augmented metamodel $Q_1(M)$ by replacing arguments of Q_3 (classes Commut.Vehicle and LeisureVehicle) by their images in $Q_1(M)$ along mapping T_1 as shown in the figure. In this way, mapping T_1 can be homomorphically extended to mapping $Q_3(T_1)$, and now mappings T_3 and $Q_3(T_1)$ can be composed. The lower part of Fig. 8 is

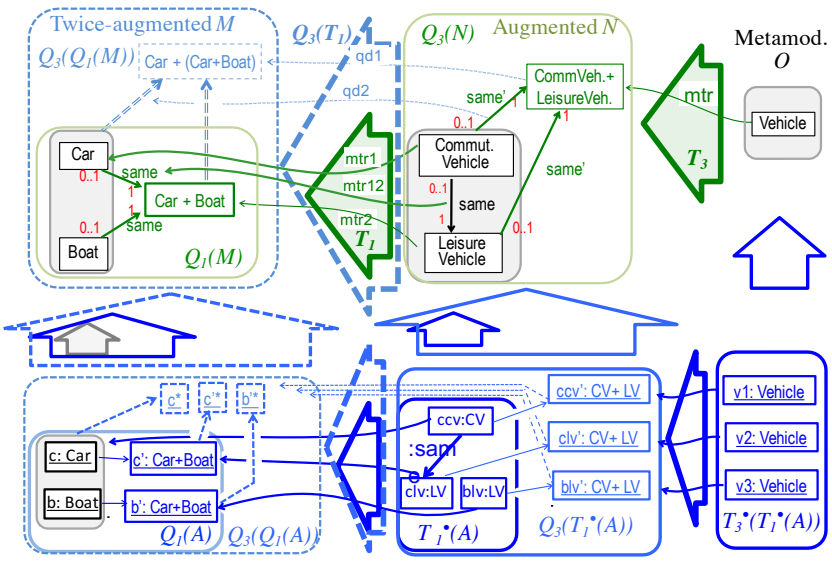


Fig. 8: Model chaining via mapping composition

a "link-full" demonstration that the consecutive composition of two executions is equal to the execution of the composed mapping:

$$T_3^{\bullet}(T_1^{\bullet}(A)) = (T_3 \circ T_1)^{\bullet}(A) \text{ and } T_3{}^{\blacktriangleleft}(T_1^{\bullet}(A)) = (T_3 \circ T_1)^{\blacktriangleleft}(A),$$

where $\circ$ denotes sequential mapping composition. In fact, the case is nothing but an instance of the well known construction of query substitution and view compositionality: a view of a view is a view of the very first source.

5 From rule-based MMT to Kleisli mappings (and back)

General landscape. Rule-based programs such as in ATL or ETL are good for effective execution of MMTs, but their analysis and manipulation may be difficult. For example, it is not easy to say whether the result of transformation always satisfies the constraints of the target metamodel, and chaining rule-based transformations is difficult too. The Kleisli mapping encoding of MMTs can help by providing a more abstract view that may be better amenable for some analyses and operations. For instance, it was shown in [6] the Kleisli mapping encoding converts the target metamodel conformance into to a well-known logical problem of whether a formula is entailed by a theory, for which a standard theorem prover or a model checker could be applied. To perform such logical analyses, we need a translating procedure $\mathsf{tl2km}\colon \mathsf{TL} \to \mathsf{KM}_{\mathsf{QL}}$, where TL denotes the set of programs written in the rule-based language at hand, and $\mathsf{KM}_{\mathsf{QL}}$ denotes the set of Kleisli mappings over some query language QL. Below we will omit the subindex QL.

To employ the KM approach for operations over rule-based transformations, we also need an inverse translation $\mathsf{km2tl}\colon \mathsf{TL} \leftarrow \mathsf{KM}$. Suppose that $\mathbf{T}_1\colon M \to N$ is a rule-based transformation that maps instances of metamodel M to instances of metamodel N, and $\mathbf{T}_3\colon N \to O$ is another transformation that we want to chain with $\mathbf{T}_1$. To do this, we translate both transformations to the respective Kleisli mappings T_1, T_3, then perform their sequential composition and obtain mapping $TT = T_3 \circ T_1$ as it was explained in Sect. 4, and then translate the result back to TL obtaining a rule-based transformation $\mathbf{T} = \mathsf{km2tl}(TT)$, which is semantically equivalent to sequential composition of $\mathbf{T}_1$ and $\mathbf{T}_3$. In a similar way, we can perform Boolean operations over rule-based parallel transformations $\mathbf{T}_i\colon M \to N$, $i = 1, 2$ (see Sect. 4) and employ them, e.g., for reuse.

Listing 1.1: $\mathbf{T}_1$ expressed in ATL

```
module T1;
create OUT : N from IN : M;
rule car2vehicle {
  from c  : M!Car
  to   cv : N!CommutVehicle
            (same <- lv),
       lv : N!LeisureVehicle
}
rule boat2vehicle {
  from  b : M!Boat
  to   lv : N!LeisureVehicle
}
```

Listing 1.2: $\mathbf{T}_3$ expressed in ATL

```
module T3;
create OUT : O from IN : N;
rule commutVehicle2vehicle {
  from  cv : N!CommutVehicle
  to     v : O!Vehicle
}
rule leisureVehicle2vehicle {
  from lv : N!LeisureVehicle
  to    v : O!Vehicle
}
```

Listing 1.3: Query QQ in ATL

```
module QQ;
create OUT : QQM from IN : M;
rule car2carcarboat {
  from  c : M!Car
  to   qqc : QQM!Car
             (same <- qqccb1),
             (same <- qqccb2),
   qqccb1 : QQM!CarCarBoat
   qqccb2 : QQM!CarCarBoat
}
rule boat2carcarboat {
  from  b : M!Boat
  to   qqb : QQM!Boat
             (same <- qqccb),
    qqccb : QQM!CarCarBoat
}
```

Listing 1.4: Mapping m_{TT} in ATL

```
module mTT;
create OUT : O from IN : QQM;
rule carcarboat2vehicle {
  from ccb : QQM!CarCarBoat
  to     v : O!Vehicle
}
```

Example. Suppose we want to chain two ATL transformations: $\mathbf{T}_1$: $M \to N$ and $\mathbf{T}_3$: $N \to O$ specified in Listings 1.1 and 1.2 resp. The first one is an ATL encoding of transformation $\mathbf{T}_1$ in Fig. 1, and the second transformation, in fact, merges two classes in N into class Vehicle in O. To chain these transformations, we first encode them as Kleisli mappings as shown in Fig. 8. Then we compose them using query substitution as explained in Sect. 4.2, and let the resulting mapping be $m_{TT}\colon QQ(M) \leftarrow O$, where $QQ(M) = Q_3(Q_1(M))$ is the composed query against metamodel M, which augments it with derived class $\mathsf{Car} + \mathsf{Car} + \mathsf{Boat}$ (see Fig. 8). Now we need to translate Kleisli mapping $TT = (QQ, m_{TT})$ into an ATL transformation.

We do the inverse translation in two separate steps. First, we translate query QQ into an ATL module `QQ` as shown in Listing 1.3. Then we translate the mapping $m_{TT}\colon QQ(M) \leftarrow O$ into an ATL module `mTT` as shown in Listing 1.4. The structure of these modules makes their chaining quite straightforward (in contrast to chaining the initial two transformation), and the result is shown in Listing 1.5. Of course, in our trivial example, chaining the initial modules is also easy, but even in moderately more complex cases, chaining ATL modules is difficult, whereas with the Kleisli mapping approach, all the complexity is managed via query substitution, while the final chaining of the query module `QQ` and the mapping module `mTT` remains simple (see Fig. 8).

Automatic translations TL ↔ KM and ATL. Finding general algorithms for automatization of both translations is a very non-trivial task because of the conceptual and technical differences between the two views of MMTs. Particularly, the TL view is *elementwise*, ie, based on model elements, while the KM—view is *setwise* as queries are typically formulated as operations over sets (cf. SQL). Bridging the gap is a challenge, and we have began to work in this direction for the case of ATL as a rule-based language. Below we argue why, we think, ATL should be sufficiently well-amenable for the TL ↔ KM translations.

Listing 1.5: TT as an ATL transformation

```
module TT;
create OUT : O from IN : M;
rule boat2vehicle {
  from  b : M!Boat
  to    v : N!Vehicle
}
rule car2vehicle {
  from c  : M!Car
  to   v1 : O!Vehicle
       v2 : O!Vehicle
}
```

In rule-based transformation languages, rules applied to different parts of the model may interact in complex ways. However, in ATL, inter-rule communication is specially constrained by specific properties of the language (cf. [7]), which make it suitable to express declarative metamodel mappings. Such properties are: ① *forbidden target navigation*, ② *locality*, ③ *non-recursive rule application*, and ④ *single assignment on target properties*.

Owing these properties, a direct correspondence between a metamodel mapping and the ATL constructs can be defined as follows. A *meta-traceability mapping* can be completely described using a *module* in which every independent *matched rule* represents a *meta-traceability link* between two *entities* (e.g. classes) of the source and the target metamodels ①. This is possible due to the fact that a *matched rule* is the only responsible of the computation of the elements it creates ②, and model elements that are produced by ATL rules are not subject to further matches ③. In such a *matched rule*, the *source* of the link is represented using the *to* block, and the *target* of the link is represented using the *from* block. On the other hand, in the case of *meta-traceability links* between two properties of the source and the target metamodel, the *source* of the link is represented by the property being initialised, and the *target* of the link is represented by the property on the source metamodel. This assignment is possible because ATL allows assigning the default target model element of another rule. In this case, *meta-traceability links* between two properties can only be defined in the rule that maps the owner of the property, since that rule is the only responsible of the initialisation of the attributes of the owner ②, and the assignment of a single-valued property in a target model element happens only once in the transformation execution ④.

6 Related work

Traceability understood broadly is an enormous area [1,15], but in the present paper we consider its special sub-area connected with MMT, and especially using traceability mappings as transformation definitions.

Atlas Model Weaver was proposed in [8] as a means to facilitate transformation design. *Weaving models (WMs)* represent different kinds of relationships between model elements, and are comparable to the metamodel mappings considered in our paper. WMs are executed with higher-order transformations, which build an ATL transformation from a WM. However, WMs aim to manage MMT's complexity in a single step. Hence, as WMs cannot cope with all the semantics provided by general purpose transformation languages, they may produce incomplete ATL transformations that must be manually reviewed.

To provide a more structured framework to define executable mappings between metamodels, Wimmer et al. [19] propose a set of *kernel* operators, from which *composite mapping operators* are built. The building process is performed by connecting input and output ports. Executability of the composite complex mappings is achieved by extending the framework proposed in [8].

Since meta-traceability links provide limited semantics compared to generic MMT, and building rich mappings may imply complex meta-traceability links, a

generic transformation algorithm is proposed in [9] to execute mapping models. This proposal uses simple mapping models to execute MMT putting the hard work in the execution of the algorithm. Any ambiguity caused by the semantic gap between mapping models and MMT is solved by using a "smart" algorithm that analyses the target metamodel. Since the philosophy of this proposal is to provide a result *as good as possible*, it does not guarantee that ambiguities are always correctly resolved (and may even require users' interaction).

Paper [18] provides an in-depth discussion of traceability in the context of QVT-rules execution, and hence executability of meta-traceability links. The machinery employed is described informally, but seems close to our use of pullback. The overall picture is broader than ours and includes mapping refinement, dynamic dispatch, and concurrency. A formalization of these constructs in terms of our framework would be a useful application; we leave it for future work.

In papers [2,10,6], the authors translate the source and the target metamodels to Alloy and specify the transformation rules as relations. In terms of our paper, both queries and mappings are encoded as logical theories, whose execution is provided by Alloy instance finder. Separating queries and mappings is discussed in [4], but the expressiveness of pullback seems underestimated; particularly, the many-to-many traceability mappings are not considered. A precise formalization of traceability mappings with queries in categorical terms as Kleisli mappings is provided in [5], sequential composition then follows from Kleisli mapping composition. However, the general context for paper [5] is general inter-model relationships (which corresponds to a broad view of traceability as correspondence emphasized in [1]), while in the present paper we consider traceability in the MMT context and are focused on mapping execution. In neither of the works mentioned above, operations over transformations are considered, and we are not aware of their explicit introduction and discussion in the literature.

7 Conclusion

Technological importance of traceability mappings for model transformations is well-known, but they have often been considered as an auxiliary element generated during the transformation execution and providing accessory information. This paper argues that traceability mappings should instead be regarded as a core aspect of the transformation definition, and a key instrument in the transformation management. We have shown that MMT semantics is essentially incomplete without traceability links between models, which should be typed by the respective metalinks between metamodels. Metalinks taken together constitute a traceability mapping between metamodels, which can be executed and thus appears as a transformation definition. We considered two cases of such definitions: simple mappings whose execution can be specified by an operation called pullback, and complex (Kleisli) mappings involving queries against the source metamodel, whose execution consists of the query execution followed by pullback. An important consequence of defining transformations via mappings is that algebraic operations over mappings can be translated into operations over transformations specified in conventional transformation languages. We argue

that ATL should be well amenable to such translation, and presented a simple example illustrating these ideas. Of course, a real application of our algebraic framework requires an automatic translation from ATL to Kleisli mappings and back. This challenging task is an important future work.

References

1. N. Aizenbud-Reshef, B. T. Nolan, J. Rubin, and Y. Shaham-Gafni. Model traceability. *IBM Systems Journal*, 45(3):515–526, 2006.
2. K. Anastasakis, B. Bordbar, and J. M. Küster. Analysis of model transformations via alloy. In *Proceedings of the 4th MoDeVVa workshop Model-Driven Engineering, Verification and Validation*, pages 47–56, 2007.
3. A. Benelallam, A. Gómez, M. Tisi, and J. Cabot. Distributed model-to-model transformation with atl on mapreduce. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering*, SLE 2015, pages 37–48, New York, NY, USA, 2015. ACM.
4. M. Didonet Del Fabro and P. Valduriez. Towards the efficient development of model transformations using model weaving and matching transformations. *Software and System Modeling*, 8(3):305–324, 2009.
5. Z. Diskin. Model synchronization: Mappings, tiles, and categories. In J. M. Fernandes, R. Lämmel, J. Visser, and J. Saraiva, editors, *GTTSE*, volume 6491 of *LNCS*, pages 92–165. Springer, 2009.
6. Z. Diskin, T. Maibaum, and K. Czarnecki. Intermodeling, queries, and kleisli categories. In J. de Lara and A. Zisman, editors, *FASE*, volume 7212 of *LNCS*, pages 163–177. Springer, 2012.
7. M. Freund and A. Braune. A generic transformation algorithm to simplify the development of mapping models. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, MODELS '16, pages 284–294, New York, NY, USA, 2016. ACM.
8. L. Gammaitoni and P. Kelsen. F-alloy: An alloy based model transformation language. In *Theory and Practice of Model Transformations*, pages 166–180. Springer, 2015.
9. H. Gholizadeh, Z. Diskin, S. Kokaly, and T. Maibaum. Analysis of source-to-target model transformations in quest. In J. Dingel, S. Kokaly, L. Lucio, R. Salay, and H. Vangheluwe, editors, *Proceedings of the 4th Workshop on the Analysis of Model Transformations co-located with the 18th International Conference on Model Driven Engineering Languages and Systems (MODELS 2015), Ottawa, Canada, September 28, 2015.*, volume 1500 of *CEUR Workshop Proceedings*, pages 46–55. CEUR-WS.org, 2015.
10. H. Gholizadeh, Z. Diskin, and T. Maibaum. A query structured approach for model transformation. In J. Dingel, J. de Lara, L. Lucio, and H. Vangheluwe, editors, *Proceedings of the Workshop on Analysis of Model Transformations co-located with ACM/IEEE 17th International Conference on Model Driven Engineering Languages & Systems (MoDELS 2014), Valencia, Spain, September 29, 2014.*, volume 1277 of *CEUR Workshop Proceedings*, pages 54–63. CEUR-WS.org, 2014.
11. F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev. ATL: A model transformation tool. *Sci. Comput. Program.*, 72(1-2):31–39, 2008.
12. D. S. Kolovos, R. F. Paige, and F. Polack. The epsilon transformation language. In A. Vallecillo, J. Gray, and A. Pierantonio, editors, *Theory and Practice of Model Transformations, First International Conference, ICMT 2008, Zürich, Switzerland, July 1-2, 2008, Proceedings*, volume 5063 of *Lecture Notes in Computer Science*, pages 46–60. Springer, 2008.
13. D. Lopes, S. Hammoudi, J. Bézivin, and F. Jouault. *Mapping Specification in MDA: From Theory to Practice*, pages 253–264. Springer London, London, 2006.

14. F. Marschall and P. Braun. Model transformations for the MDA with BOTL. In *Proceedings of the Workshop on Model Driven Architecture: Foundations and Applications*, pages 25–36, 2003.
15. R. F. Paige, N. Drivalos, D. S. Kolovos, K. J. Fernandes, C. Power, G. K. Olsen, and S. Zschaler. Rigorous identification and encoding of trace-links in model-driven engineering. *Software and System Modeling*, 10(4):469–487, 2011.
16. The Eclipse Foundation. ATL, Oct., 2016. URL: `http://www.eclipse.org/atl/`.
17. The Eclipse Foundation. Epsilon, Oct., 2016. URL: `http://www.eclipse.org/epsilon/`.
18. E. Willink and N. Matragkas. QVT Traceability: What does it really mean? In *Analysis of model transformations, AMT'15*, 4th Workshop Models'15, 2015.
19. M. Wimmer, G. Kappel, A. Kusel, W. Retschitzegger, J. Schoenboeck, and W. Schwinger. Surviving the heterogeneity jungle with composite mapping operators. In *Proceedings of the Third International Conference on Theory and Practice of Model Transformations*, ICMT'10, pages 260–275, Berlin, Heidelberg, 2010. Springer-Verlag.