# Model transformation as view computation*

**Technical Report CSRG-582**
**Department of Computer Science,**
**University of Toronto, 2008**

Zinovy Diskin

zdiskin@cs.toronto.edu

**Abstract.** Our central idea is to algebraically derive model transformations from mappings between the metamodels. Importantly, these mappings are allowed to map elements of one metamodel to elements that can be *derived* in the other metamodel (by posing suitable queries to it) but are *not immediately* present in it. In this way we build a formal semantics for a surprisingly wide class of model transformations. We also formally expound a number of notions usually treated informally or semi-formally, e.g., model analysis and migration, or the difference between bi-directionality and bijectiveness.

## 1 Introduction and motivating discussion

Model transformation, MT, is a key component of many activities in software engineering. In databases, it appears in data warehousing (the infamous extract/transofrm/load cycle) and in numerous scenarios of data and schema integration from federated databases to web data. In programming, if we consider code artifacts as models whose metamodels are usually set by the corresponding grammars, model transformations are everywhere: from model-based code generation to compilation to reverse engineering. In the MDA/MDD vision, programming ***is*** model transformation and software development is all about MT.

***1.1 The problem.*** Models are normally comprised of many different elements and their structural links, which in MT-tasks need to be traversed. It makes MT-programming very laborious and error-prone [1], and many specific

* Supported by Bell Canada through the Bell University Labs, NSERC, and the Ontario Centres of Excellence.

languages and tools have been designed to aid the programmer. The field is flourishing as can be seen from the surveys and experience reports [3, 17, 11, 14], and from acceptance in 2005 of an OMG standard for the area, “Queries, views, transformations (QVT)” [15]. However, the modern MT-languages (though computationally powerful or even Turing complete) are at a low abstraction level and still require the programmer to deal with many model navigation details. In order to raise the level of abstraction, a *semantic* theory for MT is needed but has not yet been developed. Understanding semantics of MT-programs is also required for their reuse; the latter is important because of high costs of MT-program development. Tool design also needs a certain level of abstraction and genericness of the functionalities being offered, which can hardly be achieved without an underlying semantic theory. It not only makes building tools expensive, but the users cannot use them effectively without a clear semantic picture of the procedures being used.

So far, building theoretical foundations under MT has been going mainly within the graph grammars stream (GG), especially triple graph grammars (TGG) that have influenced the definition of the QVT standard (see, e.g., [8] for a TGG-outline). However, the GG-view addresses the operational rather than denotational semantics of model transformations. A GG-theory explains how to transform a (graph-based) model by applying to it a *given* set of rewriting rules but says little about how to design this set and what goals it aims to accomplish. The lack of such high-level semantics in GG-foundations for MT was emphasized in [16], which was itself aimed at filling the gap by “approaching semantics issues in ... model transformations from first principles.” The present paper is similar in its intentions but differs in their realization: we go much further in mathematical elaboration and apply graph-based (rather than string-based) logic and algebra.

***1.2 The approach.*** Our central idea is to *derive* model transformations from mappings between the metamodels. Indeed, for a given MT-context, we are normally interested in a *generic* procedure translating any model over some (source) metamodel $S$ to a model over another (target) metamodel $T$. It is reasonable to assume that $T$-elements should be already “hidden” somehow amongst $S$-elements. The simplest case is when we have a mapping $m\colon T \to S$ interpreting $T$-constructs by suitable $S$-constructs. Yet in practice we rarely have such a simple relationship between the metamodels. As a rule, $S$-counterparts of $T$-constructs are not immediately present in the set of $S$-elements but can be derived from the latter by applying suitable algebraic operation*queries* in the database jargon). The metamodel relationship is then specified by a mapping $m\colon T \to \mathsf{der}^Q S$ into some augmentation $\mathsf{der}^Q S$ of the source metamodel with derived elements (with $Q$ referring to the set of operations/queries used).

We will show that given such a mapping $m$, the result of translating $S$-models into $T$-models could be defined as the result of some algebraic (meta)operation over models and model mappings. This operation is well known in category theory under the name of *pull-back* (PB) and appears in many different contexts. Particularly, when we consider typed graphs and their transformations, the retyping procedure is given by the corresponding pull-back (see, e.g., [10]). Model transformation also can be seen as a sort of retyping. However, the crucial point is that first the source model must be augmented with new derived elements, which are then retyped according to the metamodel mapping. It is the first part of this process, which captures the intended semantics of the transformation and makes it semantically non-trivial that we will demonstrate in the paper. The second part, retyping itself, is a purely algebraic procedure automatically performed by PB. In the GG-approach, these two principally different (at least semantically) components of MT are merged in one big set of rewriting rules, which blurs the semantics of the transformation.

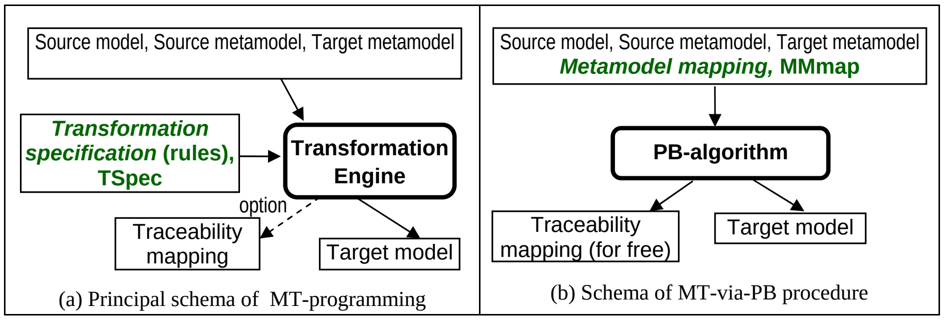

Source model, Source metamodel, Target metamodel
Transformation specification (rules), TSpec
Transformation Engine
option
Traceability mapping
Target model
(a) Principal schema of MT-programming
Source model, Source metamodel, Target metamodel
Metamodel mapping, MMmap
PB-algorithm
Traceability mapping (for free)
Target model
(b) Schema of MT-via-PB procedure

Fig. 1: Two approaches to MT: conventional (a) (adapted from [9]) and algebraic (b). Components to be specified by the programmer are shown in italic (green)

The two approaches to MT, the conventional and ours, are schematically compared in Fig. 1. The key block of the former is Transformation specification, TSpec. It says how each element, or a group of elements, in the source model are to be transformed into an element or a group in the target model. Hence, the programmer is required to create an *elementwise* specification relating the source and the target models. It is the elementwise nature of TSpec that makes MT-programming notoriously laborious and error-prone. In our algebraic approach, the transformation specification amounts to setting a metamodel mapping $m$ as above. Building the mapping is also an elementwise task but it is easier than specifying rewriting rules because (i) metamodels contain significantly fewer elements than models and (ii) it is generally easier to build queries than rewriting rules. After the mapping is built, all the rest is done automatically by the PB-algorithm. If we read Fig. 1 from right to left, we can say that the mapping between the metamodels is hidden in the set of rules TSpec. In this sense, the difference between the two approaches can be seen as yet another retelling of the classical story of software engineering: "semantics of the domain is hidden in the application code."

***1.3 Underlying mathematics and its presentation in the paper.*** Models are graph-based structures endowed with constraints. The discussion above shows that it is vital for MT to consider operations/queries over models, which augment them with derived elements. To avoid idiosyncrasies of using particular logics for specifying constraints and particular query languages for operations, we use a fairly abstract version of logic and algebra that subsumes OCL, SQL, FOL and higher-order logics as well [12]. In this logic, constraints are *diagram predicates* whose arities are graphs, and queries are *diagram operations* whose input/output arities are graphs too. We will refer to this framework as DLA - diagram logic and algebra. In the paper, we accurately describe predicate diagrams and operations that appear in our examples but do not give formal definitions as they would make the text too technical and bulky. A one-page outline of DLA basics can be found in [5], and full technical accounts are in [4, 19].[1].

Another technical apparatus heavily employed in our approach is categorical algebra (CA), especially the machinery of Kleisli mappings [13]. Our key notion – a base metamodel mapping – is nothing but a Kleisli mapping for a corresponding algebraic theory (query language). Again, we motivate and describe our constructions rather than give formal definitions. If the reader is familiar with CA, he will be able to extract such definitions from our descriptions; if not, the interested reader will be motivated to take a look at CA.

## 2 Model transformation via algebraic procedures

### 2.1 What is the input?

Suppose we want to design a procedure transforming bipartite graphs (*bp-graphs* or *nets*) modeling concurrent distributed computation into directed multi-graphs (just *graphs* in this section). The intended semantics of the transformation is to project a model over the rich metamodel onto a poorer metamodel but preserve certain aspects we consider important. Since this paper is about model transformation rather than modeling concurrency, we consider the semantic context approximately and mainly for illustration purposes.

Firstly, we must precisely define the objects we are going to manipulate: bp-graphs/nets and graphs. In MDA-parlance, this means that we need to build their graph-based metamodels. The metamodel of graphs, Graph, is well-known and presented in Fig. 1(a).

A sample net is shown in Fig. 1(b0), its metamodel BpGraph and abstract syntax are presented in, respectively, Fig. 1(b) and (b'): ignore the dashed (blue with a color display) part in the latter for the time being.[2]

We assume that what is usually called *flow relation* is presented by *in-arcs* going from places to transitions, and *out-arcs* going from transitions to places. In the metamodel, these elements are denoted by nodes iArc and oArc with correspondingly alternated source and target references, *So and *Ta, *=i,o, to

[1] Warning: in these works, semantics of predicates and operations is given in the indexed category setting, while for MT the fibred setting is more customary and convenient. Translating DLA to the fibred setting is an ongoing project at our institutions

[2] Colors help to distinguish semantically different elements. For white-black printing we use other means: dashed lines, fonts and sometimes decorate the names of elements, e.g., by prefixing them with slash.

Places and Transitions. Thus, net (b0) is represented by a typed graph shown in (b'). Note two labels [**key**] "hung" on the top and bottom arrow spans in BpGraph-graph. These labels denote two predicate declarations, [**key**](iSo,iTa) and [**key**](oTa,oSo), which say that the respective pairs of mappings uniquely determine the objects in the source nodes of the spans; such pairs of mappings are called *keys* in databases [3]. Thus, nodes iArc and oArc denote binary relations without duplication of elements.

Since we are going to transform uniformly *any* net into a graph, elements of the metamodel for graphs should be somehow found in the metamodel for nets. In other words, we should reconstruct elements of metamodel Graph within metamodel BpGraph and, in a sense, *model* nets by ordinary graphs. There are multiple ways of such modeling, which are related to semantic aspects of our future transformations. To fix semantics, suppose that we are not interested in the fine-grained structure of transitions' inputs and outputs: all that is needed is to record relations between *sets* of input and output places. In other words, we want to consider transitions as binary relations between sets of places rather than multi-ary relations over individual places. A formal realization of this idea is shown in the bottom layer of Fig. 1.

We first add to metamodel BpGraph a node **set**(Place) denoting the powerset object of Places; it is related to node Place by a binary membership relation shown with a double-dashed edge $\in$. Then we can present the two binary relations iArc and oArc by the respective mappings $\mathbf{map}(R)\colon Trans \to \mathbf{set}(Place)$ with $R$=iArc,oArc, into the powerset. These derived elements are shown with dashed (blue) lines. Our intension of modeling nets by graphs is formally fixed by a metamodel mapping $m\colon \mathsf{Graph} \to \mathsf{BpGraph}$ that interprets elements of the target metamodel by the respective elements of the source metamodel.

What we did is abstractly presented in Fig. 2(a). Nodes $S$ and $T$ are the source and target metamodels respectively. Node $\mathsf{der}^Q S$ denotes augmentation of the source metamodel with derived elements required for the interpretation. Index $Q$ refers to the set of operations (queries), which produce these elements; we will call this set the *base query*, and mapping $m$ the *base mapping* of the transformation. The vertical mapping $\sigma$ is nothing but typing of model's elements by metamodel's elements.

## 2.2 What is the output, *declaratively*

We assume that all input data required for transformation of model $M$ to the target metamodel $T$ are shown in diagram Fig. 2(a). Now we will try to figure out what should be the result of transformation according to some common sense reasons.

The first step is obvious. Derived elements of metamodel $\mathsf{der}^Q S$ are just *denotations* of necessary operations. To transform an instance $M$, we first need to compute/execute these operations for the data embodied in the instance. For example, in the metamodel $\mathsf{der}^Q\mathsf{BpGraph}$ in Fig. 1(b), two applications of

[3] and *jointly monic* in category theory

$$\begin{tikzcd}
 & & M \arrow[d, "\sigma"] \\
T \arrow[r, "m"'] & \mathsf{der}^Q S & S \arrow[l, hook']
\end{tikzcd}$$

(a) input

$$\begin{tikzcd}
\mathsf{der}^Q M \arrow[d, "\overline{\sigma}^Q"'] \arrow[dr, phantom, "\langle \mathbf{Q}\ ]/\ [\mathbf{preIm}]"] & M \arrow[l, hook'] \arrow[d, "\sigma"] \\
\mathsf{der}^Q S & S \arrow[l, hook']
\end{tikzcd}$$

(b1) query execution

$$\begin{tikzcd}
N_i \arrow[dr, "!_i"] \arrow[drr, "m_i'"] \arrow[ddr, "\tau_i"'] & & \\
 & N \arrow[r, "m'"] \arrow[d, "\tau"'] \arrow[dr, phantom, "{[=]}"] & \mathsf{der}^Q M \arrow[d, "\overline{\sigma}^Q"] \\
 & T \arrow[r, "m"'] & \mathsf{der}^Q S
\end{tikzcd}$$

(b2) model translation

$$\begin{tikzcd}
N \arrow[r, "m'"] \arrow[d, "\tau"'] \arrow[dr, phantom, "\langle \mathbf{PB}]"] & \mathsf{der}^Q M \arrow[d, "\overline{\sigma}^Q"'] \arrow[dr, phantom, "\langle \mathbf{Q}\ ]/\ [\mathbf{PB}]"] & M \arrow[l, hook'] \arrow[d, "\sigma"] \\
T \arrow[r, "m"'] & \mathsf{der}^Q S & S \arrow[l, hook']
\end{tikzcd}$$

(b3) final output

Fig. 2: MT-transformation via diagram operations

operation **map** are specified. Their execution within the instance (b') results in three new nodes of type **set**(Place) (along with membership references) and the corresponding source and target references from the transitions $f$ and $g$. Thus, executing queries gives us an augmented source model $\mathsf{der}^Q M$ typed over the augmented source metamodel $\mathsf{der}^Q S$ as shown by Fig. 2(b1), where $\overline{\sigma}^Q$ denotes the extended typing mapping. We may say that node $\mathsf{der}^Q M$ (together with two adjoint arrows) is derived by applying the operation $[\mathbf{Q}\rangle$ to the lower-right triangle of elements (note asymmetry in the brackets embracing the operation symbol).

An important feature of adding derived elements to a model is that it should not have side-effects and does not affect the input data. Formally, the pre-image $[\overline{\sigma}^Q]^{-1}(S)$ of the source metamodel must be exactly the original source model $M$, $\mathbf{preIm}_{\overline{\sigma}^Q}(S) = M$.

Now we need to transform the augmented model $\mathsf{der}^Q M$ towards the target metamodel $T$ as is prescribed by the mapping $m$. Let $N$ denotes the result of transformation we are looking for. Since it must be a model over the target metamodel $T$, it comes equipped with a typing mapping $\tau\colon N \to T$, Fig. 2(b2). In addition, it is reasonable to require that each element of the new model $N$ should be traced back to either basic or derived element of the original model. It is also reasonable to assume that this *traceability* relationship is structure-preserving and, hence, traceability appears as a morphism $m'\colon N \to M$ between the models. Moreover, if an element $a$ in model $N$ has a type $a.\tau$ and is traced back to element $a.m'$, then the type of the latter should be $a.\tau.m$. That is, $a.m'.\overline{\sigma}^Q = a.\tau.m$ for all elements $a$ in model $N$ and the square diagram in (b2) must be commutative.

The properties just listed do not necessarily characterize a unique model and we can well imagine another model $N_1$ along with mappings $\tau_1$ and $m_1'$ making commutative square as required. What should distinguish the desired translation $N$ among other possible translations $N_1$, $N_2$, ... is that the former must not lose information and hence be maximal amongst all models $N_i$ in some sense. A generic way of specifying such maximality is to require the existence of a unique mapping $!_i$ to the model $N$ from any candidate model $N_i$ (with mappings $\tau_i$ and $m'_i$ making the outer square diagram in Fig. 2(b2) commutative) such that both triangle diagrams are commutative too. Following the category theory terminology, we will call the property just described *universal*.

Thus, we define the transformation of model $(\mathsf{der}^Q M, \overline{\sigma}^Q)$ to a metamodel $T$ determined by mapping $m$ to be an operation that produces the left square in Fig. 2(b2), which is (i) commutative and (ii) possesses the universal property above. The latter implies that the result of transformation is inique up to isomorphism between the models and hence model transformation is indeed an operation. This operation is well-known in category theory and is called *pull-back* (PB). What we have done above is then a motivation to identify the informal notion of model transformation with a formal algebraic procedure called PB. In addition, it can be easily checked that the preImage operation on graphs is also nothing but a pull-back. Then the final result of model transformation can be elegantly specified as shown in diagram (b3). The label $\langle\mathbf{PB}]$ in the left square denotes an operation (note asymmetric bracketing $\langle,]$) while label $[\mathbf{PB}]$ in the right square denotes a predicate specifying a postcondition for the operation $\langle\mathbf{Q}]$ (and bracketing is symmetric).

## 2.3 What is the output, *constructively*

The definition of model transformation given above is entirely declarative and says nothing about *how* to compute the transform. It would be basically useless unless a group of mathematical results, which show how to compute PBs for various universes of objects and mappings between them. Particularly, a well-known result of category theory says that if a universe has Cartesian products and some simple set-comprehension schema determined by equalities, then the pull-back object $N$ can be computed as follows:

$$N = \left\{ (a,b) \in \mathsf{der}^Q M \times T \,\middle|\, \overline{\sigma}^Q(a) = m(b) \right\}, \qquad (1)$$

that is, $N$ is a relation over $\mathsf{der}^Q M$ and $T$ with $m'$ and $\tau$ being the projection mappings.

Moreover, if the objects of the universe are systems of sets (e.g., a graph is a set of nodes and a set of arrows) and mappings between them are mappings between these sets (sending nodes to nodes and arrows to arrows), then PBs can be computed by applying definition (1) to each of the component sets.[4] For instance, if our universe consists of graphs and their mappings, PBs can be computed by applying the definition (1) twice: for nodes and for arrows. Example in Fig. 1 shows how it works. The graph (a') is computed by the direct application of the definition (1): elements of this graph are pairs of elements from graphs (a) and (b') with pairing denoted by an infix bullet. The trace mapping is

[4] We have omitted some important details, they can be found in any textbook on category theory considering presheaf toposes.

given by the first component of the pairs and typing by the second one. Diagram (a") shows the object (a') in the familiar concrete syntax.

Note that the result provided by the PB-operation conforms to our expectations. Indeed, the idea behind the base mapping (ab) is to interpret Nodes by sets of Places disregarding the internal structure of these sets. Correspondingly, the result of transformation, (a'), is a specific projection of net (b0), in which the fine-grained picture of place distribution is ignored. We may say that semantics of transformation is encoded by the base mapping (ab) while PB just routinely explicates this semantics by an accurate retyping of the elements involved. We consider this pattern to be an important advantage of the approach: it makes semantics explicit and localizes it in the base mapping, which in its turn facilitates understanding and reuse. Below we will consider more examples of how it works (section 3.2). In addition, more complex and practically interesting examples can be found in [7][6], where reverse engineering of SQL-schemas into ER-diagrams and translation of (a sort of) UML class diagrams into SQL are shown to be also provided by the pattern we considered. For further references, we will call it *(Q+PB)-pattern*.

**Remark (Constraints).** Normally, metamodels are graph-based structures endowed with conditions that constrain the set of their instances: to be a valid model, the typing mapping $\sigma\colon M \to S$ must satisfy these constraints. An important result of the diagram logic says that if $m$ is compatible with the constraints embodied in metamodels $T$ and $S$, then as soon as the source model $(M, \sigma)$ satisfies the constraints in $S$, its PB-transform $(N, \tau) = PB(M, \sigma, m)$ satisfies the constraints in $T$. Thus, the PB-operation transforms valid models into valid models.

## 3 Model transformation as view materialization: Flow of information during model transformation.

Our work in the previous section can be summarized by the slogan "MT = Q+PB". It was shown in [6] that a major database construct of *view* consisting of view definition and view materialization (VM) is also specified by the (Q+PB)-pattern: the base mapping is a view definition (metamodels are schemas in the database terms), and the PB-procedure materializes it (models are data). We conclude that mathematically "MT = VM". The pragmatic contexts are different but the common formal base provides some useful analogies and ideas.[5] Particularly, we will see below that the fundamental notion of bi-directionality of model transformation actually subsumes two different problems: view updatability and view invertibility. This point seems to be underestimated or missed from the literature.

[5] The more so that VM have been studied by the database community since seventies while MT is a relatively novel research area.

## 3.1 A bit more of abstraction.

It will be useful for our discussion to present the (Q+PB)-pattern in more abstract terms focusing on the global mapping from the class of all source models to the class of target models. Figure Fig. 3 shows two consecutive abstractions of our MT-semantics developed in Fig. 2. Diagram (b4) is an abbreviation of diagram (b3), where $v, v'$ encode the pairs $(Q, m)$, $(Q, m')$ respectively. The arrow defines a *view* to metamodel $S$, and we will call it a *view mapping.* The double-bracketed label [**PB**] denotes the composition of operation [**Q**⟩;[**PB**⟩, and thus defines $(N, \tau)$ to be the materialization of view $v$ to the model $M$; we will also call it the *$v$-reduct* of $M$.

Given a view $v\colon\ T \Rightarrow S$, *any* model $(M, \sigma)$ over the source metamodel can be transformed into a model $(N, \tau)$ over the target metamodel. Hence, a view mapping $v$ gives rise to a mapping between the sets of instances $v^{\#} : \boldsymbol{inst}(S) \to \boldsymbol{inst}(T)$ as shown by diagram (b5), note that the direction of mapping is reversed. Also note that while view $v$ maps model's elements to model's elements, mapping $v^{\#}$ maps whole models to whole models. We will sometimes call such mappings *functorial* or *functors.*

$$\begin{tikzcd} N \arrow[r, Rightarrow, "v'"] \arrow[d, "\tau"'] \arrow[dr, phantom, "\langle\langle\ \mathbf{PB}\ ]]"] & M \arrow[d, "\sigma"] \\ T \arrow[r, Rightarrow, "v"] & S \end{tikzcd}$$

(b4): abstract version of (b3)

$$\begin{tikzcd} \boldsymbol{inst}(T) & & \boldsymbol{inst}(S) \arrow[ll, "v^{\#}"'] \\ T \arrow[rr, Rightarrow, "v"'] \arrow[rr, phantom, ""{name=A}] & & S \arrow[from=A, to=1-2, mapsto] \end{tikzcd}$$

(b5) even more abstract version

$$\begin{tikzcd} \boldsymbol{inst}(T) \arrow[r, shift right, "w^{\#}"'] \arrow[r, phantom, "{[\mathbf{inv}]}"] & \boldsymbol{inst}(S) \arrow[l, shift right, "v^{\#}"'] \\ T \arrow[r, Rightarrow, shift left, "v"] \arrow[r, phantom, "{[[\mathbf{inv}]]}"] & S \arrow[l, Rightarrow, shift left, "w"] \end{tikzcd}$$

(c) invertibility

Fig. 3: (Continues Fig. 2). From metamodel mappings to functors between sets of models: $v$ is a view mapping, $v'$ is the model trace mapping and $v^{\#}$ is the model transformation (meta)mapping or functor.

## 3.2 View updatability and MT.

Database views are designed to select a specific part of data stored in the database. They are thus projections or *reducts* of the database. Formally it means that the functor $v^{\#}$ is *not* supposed to be injective and the set $[v^{\#}]^{-1}(N)$ includes many models over $S$ besides $M$. It implies that propagating view updates back to the source can be problematic: if the user of the view modifies $N$ into a model $N'$, it's not clear in general which of the models $[v^{\#}]^{-1}(N')$ should be taken to be the updated version of $M$. Moreover, the modified model $N'$ can go beyond the range $\{v^{\#}(M) \mid M \in \boldsymbol{inst}(S)\}$ of legitimate views, if the functor $v^{\#}$ is not surjective. These anomalies constitute the *view update problem* that attracted much research in the database area, see [2] for a discussion and references.

The MT area sets another context but surely many of model transformations aim at abstracting/filtering the information and thus are views: typical examples are reverse engineering and model analysis. Table 2 presents a few samples illustrating the issue.

The table presents three different generic transformations of bp-graphs into graphs: each one is set by a certain semantic context and is specified by the corresponding metamodel mapping (see Fig. 2).

The first transformation is mainly syntactical and simply ignores the difference between places and transitions. To realize this as a view mapping, we first augment the source metamodel with two abstract classes: Node and Arc together with two abstract associations /so and /ta between them as specified in the leftmost column of Query Definition Table. Then we build mapping $v1$ of the target metamodel to the augmented source metamodel as shown in the right part of Fig. 2(a). The result of the transformation determined by this mapping is shown in cell (b1): the PB-operation produced what we expected. Since a lot of information is lost with this transformation, the view update anomalies appear immediately. For example, the user of view (b1) can draw an arc between nodes M and K or$f$ and $g$, which does not make sense for the source net.

Base mapping $v2$ is the same that was used in our example in Table 1 (up to inessential renaming of derived elements): the fine-grained distribution of places on the inputs and outputs of transitions is ignored. In contrast to the source model in Table 1, the source model (b0) in Fig. 2 is asymmetric and the results of forgetting the distribution are even more visible. For example, the transformed graph (b2) shows a new node J, whose relationship to nodes K and L is entirely lost. Then a seemingly local update, say, deletion of arrow $f$ would cause deletion of the entire graph (b2), or drawing an arrows from J to K cannot be propagated back to net (b0).

The idea behind the third transformation is to ignore the concurrent nature of net transitions and consider them as sets of independent single-source/single-target graph transitions. To this end, given a transition $f$, we compose each in-arc with each out-arc and replace $f$ with pairs of compatible in-out arcs. Node 2Arc denotes the set of all such composable pairs with the naturally defined source and target projections (see the right column of the Query Definition Table). Given the base mapping $v3$, application of PB to source model (b0) produces graph (b3). Single in-out transitions $g$ and $h$ are losslessly transformed while transition $f$ is split into arrows $f1$ and $f2$. The latter are executed concurrently but this information in lost in graph (b3).

To summarize (we return to diagram (b5) in Fig. 3): in certain special cases it may be possible to make a choice of a single $S$-model $M$ in the set $[v^{\#}]^{-1}(N')$ by a specially designed policy. Yet in general the view update problem is difficult to manage in a generic way, and much research is needed here. Surprisingly, this aspect of MT has gained little attention in the MT-literature and has never been discussed in precise terms. In addition, the view update problems in MT are often mixed with *view invertibility*, which we will show in the next section is a different issue.

## 3.3 View invertibility.

The problem we are going to consider is usually discussed in the MT-literature under the title of bi-directionality. However, there is a subtle but important technical detail that is usually missed by the literature. Suppose that the functor $v^{\#}$ is injective, even bijective, and hence there is the inverse functor
$[v^{\#}]^{-1}\colon \boldsymbol{inst}(T) \to \boldsymbol{inst}(S)$. Yet this functor does *not* do the job of the inverse transformation, if we want the latter to be generic, i.e., to work uniformly for all $T$-models. To ensure genericness of the inverse transformation, we need to require existence of an inverse view mapping $w\colon\ S \Rightarrow T$ s.t. $w^{\#} = [v^{\#}]^{-1}$. Thus, what we really need for generic bi-directionality is a pair of mutually inverse view mappings as shown by diagram Fig. 3(c).

In this diagram, the upper label [**inv**] denotes the predicate of being mutually inverse for ordinary mappings between sets. The lower label [[**inv**]] denotes a more complex notion. We remind that double-arrows are abbreviations of views, say,

$$v\colon T \to \mathsf{der}^Q S \text{ and } w\colon S \to \mathsf{der}^R T \tag{2}$$

with $Q, R$ denoting the corresponding (sets of) queries. To consider invertibility of these two constructs we first need to extend appropriately the domains of the view mappings and consider their *homomorphic* extensions:

$$\mathsf{der}^R v\colon \mathsf{der}^R T \to \mathsf{der}^R\mathsf{der}^Q S \text{ and } \mathsf{der}^Q w\colon \mathsf{der}^Q S \to \mathsf{der}^Q\mathsf{der}^R T. \tag{3}$$

It is seen that the crucial component of views' invertibility is invertibility of the corresponding queries. The diagram (4) shows a query $R$ to be executed after query $Q$. If the final result, the model $\overline{\sigma}^{Q;R}\colon \mathsf{der}^{Q;R}M \to \mathsf{der}^{Q;R}S$ with $\mathsf{der}^R\mathsf{der}^Q X \cong \mathsf{der}^{Q;R}X$, $X = M, S$, is canonically isomorphic (up to renaming the derived elements) to the source model $\sigma\colon M \to S$ for any model $(M, \sigma)$, we say that query $R$ is a *(right) inverse* to query $Q$ and write $Q; R = id$. If both $Q; R = id$ and $R; Q = id$, we say that queries are *mutually inverse.*

$$\begin{tikzcd}
\mathsf{der}^{Q;R}M \arrow[d, "\overline{\sigma}^{Q;R}"'] \arrow[dr, phantom, "{[\,\equiv\,]}"] & \mathsf{der}^R\mathsf{der}^Q M \arrow[l, "\cong"'] \arrow[d, "\overline{\overline{\sigma}^{Q}}^{R}"'] \arrow[dr, phantom, "{\langle\,\mathbf{R}\,]}"] & \mathsf{der}^Q M \arrow[l, hook] \arrow[d, "\overline{\sigma}^{Q}"'] \arrow[dr, phantom, "{\langle\,\mathbf{Q}\,]}"] & M \arrow[l, hook] \arrow[d, "\sigma"] \\
\mathsf{der}^{Q;R}S & \mathsf{der}^R\mathsf{der}^Q S \arrow[l, "\cong"'] & \mathsf{der}^Q S \arrow[l, hook] & S \arrow[l, hook]
\end{tikzcd} \tag{4}$$

**Definition 1 (view invertibility).** Two views $v$ and $w$ as above in (2) are called mutually *inverse* if (i) queries $Q$ and $R$ are mutually inverse and hence $\mathsf{der}^{Q;R}S \cong S$, and (ii) each of the pairs of mappings $(v, \mathsf{der}^Q w)$ and $(w, \mathsf{der}^R v)$ is mutually inverse too.

**Definition 2.** Two metamodels $S, T$ are said to possess *equal information capacity* if there is a pair of mutually inverse views between them. If $(v, w)$ is such a pair, we write $v : T \simeq S$ or, equivalently, $w : S \simeq T$.

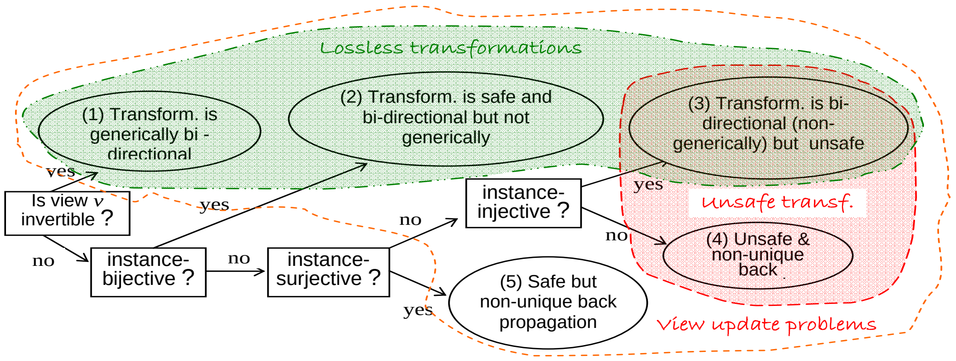
**Proposition 3.** If two metamodels have equal information capacity via an invertible view $v : T \simeq S$, their sets of instances (models) are canonically isomorphic via mutually inverse mappings $v^{\#} : \boldsymbol{inst}(S) \to \boldsymbol{inst}(T)$ and $w^{\#} : \boldsymbol{inst}(T) \to \boldsymbol{inst}(S)$.

An example of two metamodels with equal information capacities can be found in Appendix.

## 3.4 Towards formal taxonomy of model transformations.

**Definition 4.** Let $v\colon T \Rightarrow S$ be a view mapping between the metamodels and $v^{\#} : \boldsymbol{inst}(S) \to \boldsymbol{inst}(T)$ is its model transformation functor. We call $v$ (i) *safe* or *instance-surjective* if $v^{\#}$ is surjective, (ii) *precise* or *instance-injective* if $v^{\#}$ is injective, and (iii) *safe & precise* or *instance-bijective* if $v^{\#}$ is bijective.

Having $v$ instance-bijective does not imply $v$'s invertibility: instance-bijectiveness means that each $T$-model is a $v$-transform of a unique $S$-model but this correspondence may be not expressible syntactically via a suitable view. In other words, the inverse translation is not generic. In [16] the issue is discussed in detail but in less precise terms.
Definition 5 gives rise to a formal taxonomy of model transformations presented in Fig. 4.



Lossless transformations
(1) Transform. is generically bi - directional
(2) Transform. is safe and bi-directional but not generically
(3) Transform. is bi-directional (non-generically) but unsafe
Unsafe transf.
Is view v invertible ?
yes
no
instance-bijective ?
yes
no
instance-surjective ?
no
instance-injective ?
yes
no
(4) Unsafe & non-unique back
yes
(5) Safe but non-unique back propagation
View update problems

Fig. 4: View-based taxonomy of model transformation

The taxonomy shows five types of MT (final oval nodes), which can be further categorized into three pairs of complementary classes: a transformation can be (i) reductive, i.e., losing information, or lossless, (ii) safe or unsafe, and (iii) invertible or with view update problems. These classes are shown in Fig. 4 as areas with dashed boundaries. It appears that often cited bi-directionality of transformations is a loose term, whose meaning ranges over three types (1..3) of lossless transformations depending on the context.

It is interesting to relate taxonomies of MT developed in surveys [14][18] with our formal classification. Reverse engineering and model analysis appear to be reductive transformations from the group (4..5). Model migration, and various sorts of normalization, refactoring and optimization should go to the group of lossless transformations (1..3). If we want these transformations to be generically invertible, they must go to the group (1).

Finally, such transformations as model synthesis, code generation and compilation do not match the MT=VM pattern (and are thus not covered by our taxonomy), because they involve creation of new data. A precise algebraic semantics for them is a work in progress.

## References

[1] P. Bernstein and S. Melnik. Model management 2.0: manipulating richer mappings. In *SIGMOD Conference*, pages 1–12, 2007.

[2] Aaron Bohannon, Benjamin C. Pierce, and Jeffrey A. Vaughan. Relational lenses: a language for updatable views. In *PODS*, 2006.

[3] K. Czarnecki and S. Helsen. Classification of model transformation approaches. In K. Czarnecki, editor, *2nd OOPSLA03 Workshop on Generative Techniques in the Context of MDA*, 2003.

[4] Z. Diskin. Databases as diagram algebras: Specifying queries and views via the graph-based logic of skethes. Technical Report 9602, Frame Inform Systems, Riga, Latvia, 1996. http://citeseer.ist.psu.edu/116057.html.

[5] Z. Diskin. Towards algebraic graph-based model theory for computer science. *Bulletin of Symbolic Logic*, 3:144–145, 1997.

[6] Z. Diskin. Mathemtics of generic specifications for model management. In *Encyclopedia of Database Technologies and Applications*. Idea Group, 2005.

[7] Z. Diskin and J. Dingel. A metamodel independent framework for model transformation: Towards generic model management patterns in reverse engineering . In *ATEM-2006, 3rd Int.Workshop on Metamodels, Schemas, Grammas and Ontologies for reverse engineering*, 2006.

[8] H. Ehrig, K. Ehrig, C. Ermel, F. Hermann, and G. Taentzer. Information preserving bidirectional model transformations. In *FASE*, 2007.

[9] K. Ehrig, E.Guerra, J. de Lara, L. Lengyel, T.Levendovszky, U.Prange, G.Taentzer, D.Varró, and S.Varró-Gyapay. Model transformation by graph transformation: A comparative study. In *MTiP 2005, Int.Workshop on Model Transformations in Practice (Satellite Event of MoDELS 2005)*, 2005.

[10] M. Grosse-Rhode, F. Presicce, and M. Simeoni. Formal software specification with refinements and modules of typed graph transformation systems. *J. Comput. Syst. Sci.*, 64(2):171–218, 2002.

[11] R. Holt, A. Schürr, S. Elliott Sim, and A. Winter. GXL: A graph-based standard exchange format for reengineering. *Science of Computer Programming*, 60(2):149–170, 4 2006.

[12] M. Makkai. Generalized sketches as a framework for completeness theorems. *Journal of Pure and Applied Algebra*, 115:49–79, 179–212, 214–274, 1997.

[13] E. Manes. *Algebraic Theories.* No.26 in Graduate Text in Mathmetics. Springer Verlag, 1976.

[14] T. Mens and P. Van Gorp. A taxonomy of model transformation. *ENTCS*, 152, 2006.

[15] OMG, http://www.omg.org/docs/ptc. MOF QVT Final Adopted Specification. Formal/05-11-01, 2005.

[16] P. Stevens. Bidirectional model transformation in QVT: Semantic issues and open questions. In *Models-07*. Springer LNCS#4735, 2007.

[17] Laurence Tratt. Model transformations and tool integration. *Software and System Modeling*, 4(2):112–122, 2005.

[18] E. Visser. A survey of strategies in program transformation systems. *ENTCS*, 57, 2001.

[19] U. Wolter and Z. Diskin. The Next One Hundred Diagrammatic Specification Techniques – A Gentle Introduction to Generalized Sketches. Technical Report 358, Department of Informatiks, University of Bergen, Norway, 2007.

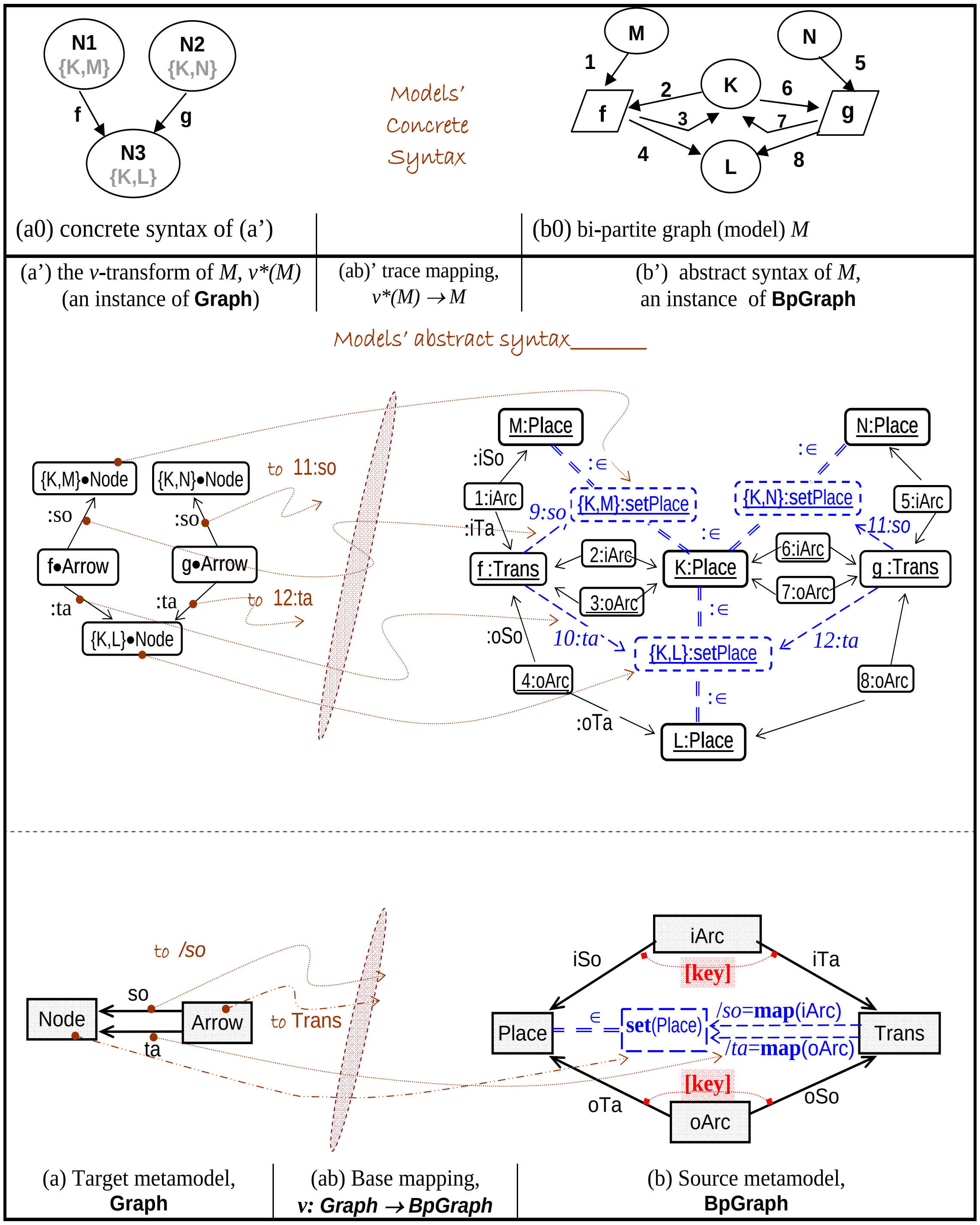
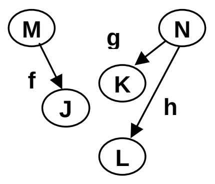
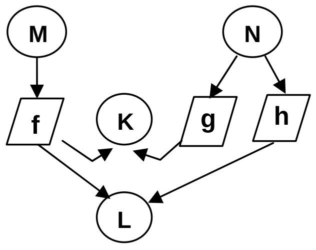


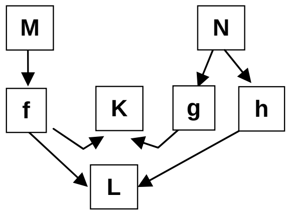
Table 1: Example of how the pull-back operation works



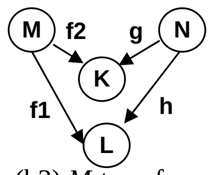
(b2) *M*-transform via *v2*



(b0) an instance *M* of **BpGraph** (circles are places, diamonds - transition)

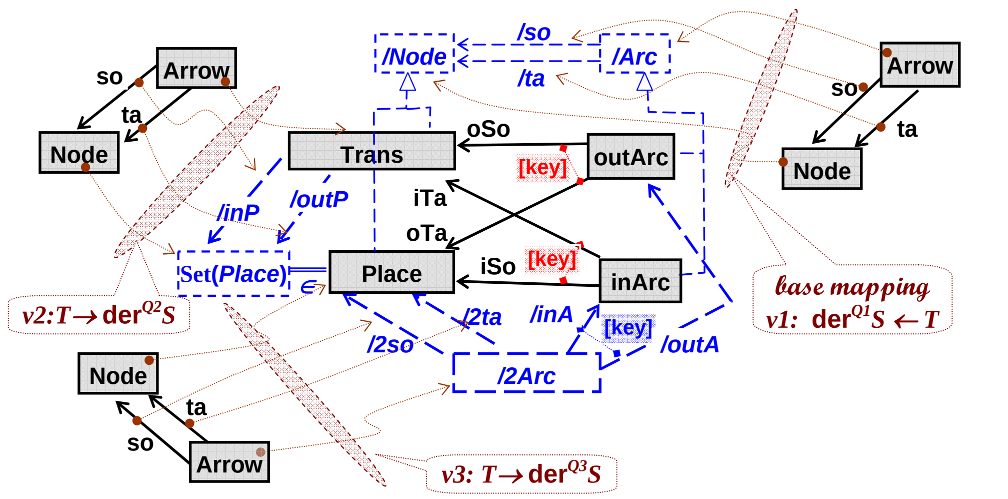


(b1) *M*-transform via *v1*, *v1*(M)*



(b3) *M*-transform via *v3*

(b) …and three corresponding model transformations via **PB**



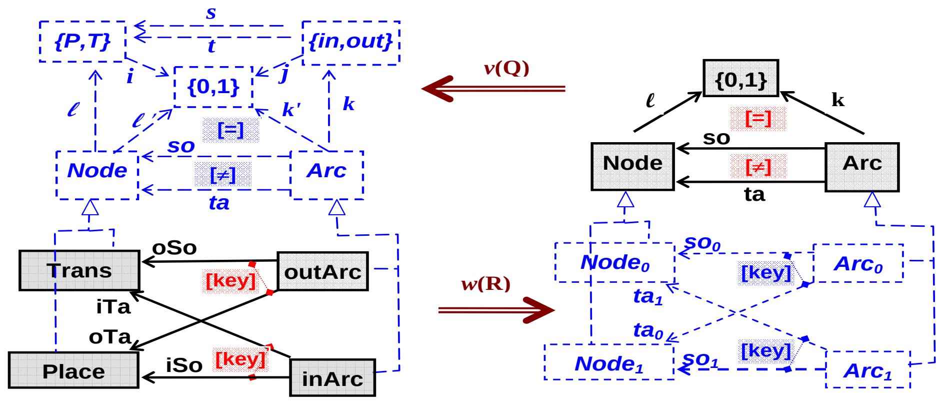
**Definitions of derived elements**

| Query $Q1=Q_a\,;\,Q_b$ with $Q_i=Q_{i1}\|\|\,Q_{i2},\ i=a,b$ | | Query $Q2=Q_a\|\|Q_b$ | | Query $Q3=Q_a;\,Q_b$ with $Q_q=Q_{a1}\|\|\,Q_{a2}\|\|Q_{a3},\ \ Q_b=Q_{b1}\|\|\,Q_{b2}$ | |
|---|---|---|---|---|---|
| (a) | *Node* = Place U Trans<br>*Arc* =inArc U outArc | (a)<br>(b) | inP=map(inArc)<br>outP = map(outArc) | (a) | 2Arc={$(a,b)\in$ inArc **x** outArc**:** iTa$(a)$= oSo$(b)$}<br>$(a,b)$.*inA* = *a*, $(a,b)$.*outA*=*b* |
| (b) | *so* = iSo U oSo<br>*ta* = iTa U oTa | | | (b) | *2so* = *inA* ; iSo<br>*2ta* = *outA* ; oTa |

(a) Three metamodel mappings…

Table 2: Transforming nets to graphs

## A Appendix. Two metamodels with equal information capacities



| **Definitions of derived elements**<br>Query Q = *Q1*;…;*Q4* with *Qi*=$Q_{i1}$‖ $Q_{i2}$‖… | |
|---|---|
| (1) | *(Node, l)* = Place \|_\| Trans<br>*(Arc, k)* = inArc \|_\| outArc |
| (2) | *so* = iSo U oSo, ***s****(in)=P*, ***s****(out)=T*<br>ta = iTa U oTa, ***t****(in)=T*, ***t****(out)=P* |
| (3) | ***i****(P)*=0, ***i****(T)*=1, ***j****(in)*=0, ***j****(out)*=1 |
| (4) | ***l'= l;i*** ***k'=k;j*** |

**BpGraph**

| **Definitions of derived elements**<br>Query R = *R1*; *R2* with *Ri*=$R_{i1}$‖$R_{i2}$ | |
|---|---|
| (1) | $Node_{0/1}$ ={*n*∈ Node: *n*.ℓ = 0/1}<br>$Arc_{0/1}$ = {*a*∈ Arc: *a*.k = 0/1} |
| (2) | *a*. $so_0$ =*a.so*, *a*.$ta_0$=*a.ta* for *a* ∈$Arc_0$<br>*a*. $so_1$ =*a.ta*, *a*.$ta_1$=*a.so* for *a* ∈$Arc_1$ |

**LblGraph**

Fig. 5: Two metamodels of equal information capacity