

Mathematics of Generic Specifications for Model Management, I

Zinovy Diskin¹

SWD Factory, Latvia and Universal Information Technology Consulting, USA

This article (further referred to as Math-I), and the next one (further referred to as Math-II, see p. 359), form a mathematical companion to the article in this encyclopedia on Generic Model Management (further referred to as GenMMt, see p.258). Articles Math-I and II present the basics of the arrow diagram machinery that provides model management with truly generic specifications. Particularly, it allows us to build a generic pattern for heterogeneous data and schema transformation, which is presented in Math-II for the first time in the literature.

INTRODUCTION

Generic MMt (gMMt) is a novel view on metadata-centered problems manifested by Bernstein, Halevy, and Pottinger (2000). The main idea is to implement an environment where one could manipulate models as holistic entities irrespectively of their internal structure. It can be done only within an integral framework for abstract generic specifications of relations between models and operations with models. However, building truly generic specifications presents a problem of a new kind for the DB community, where such familiar tools as first-order logic or relational algebra cannot help. Amongst the most pressing gMMt problems listed in Bernstein (2003)—the most complete presentation of gMMt agenda to date—almost all are *specification* problems.

Fortunately, the community does not have to develop the desired framework from scratch. As is manifested in *GenMMt*, appropriate methodology and specification techniques are already developed in mathematical category theory (CT) and waiting to be adapted to gMMt needs. The goal of *Math-I* and *Math-II* is to outline foundations of the categorical approach to MMt and demonstrate how it works in a few simple examples.

Our plan is as follows. The next section of the article describes *Kleisly arrows*—the main vehicle of the approach. The section following it presents a simple example of model merge as a sample demonstrating how the categorical treatment of MMt procedures works. We begin by representing models in a graph-based format called *sketch* and describe a general pattern for sketch merging. After that, we reformulate the pattern in abstract terms to

make it applicable to any sort of models, not necessarily sketches, although models are still supposed to be similar (be instances of the same metamodel). The last section summarizes this work by describing a generic arrow framework for *homogeneous* MMt.

The next step—managing models’ heterogeneity, particularly data and schema translation—is a highly nontrivial issue, a truly generic solution to which is often considered to be impossible (Bernstein, 2003). Nevertheless, it does exist and is presented (for the first time in the literature) in *Math-II*. Based on some category theory ideas, data and schema translation is specified in an abstract generic way via the so-called *Grothendieck* mappings. After that, homogeneous MMt specifications presented in *Math-I* can be considered as *abbreviations* for heterogeneous specifications, where arrows are interpreted as Grothendieck mappings. Particularly, the abstract pattern of homogeneous model merge developed in *Math-I* can be applied for heterogeneous merge as well. The last section of *Math-II* summarizes the results and outlines a general mathematical framework underlying generic MMt specifications.

MODEL MAPPINGS AS KLEISLY ARROWS

Do Model Mappings Really Map?

In the literature, the term “model mapping” usually refers to a specification of correspondence between models. It was noted by many authors that a general format of such correspondence must involve derived items of the models in question. In recent surveys, such as Lenzerini (2002), it became common to describe a mapping between schemas S and T as a set of assertions of the type $Q_S \rightsquigarrow Q_T$, where Q_S and Q_T are some queries over schema S and T , respectively, and \rightsquigarrow denotes some comparison operator between query outputs. It is assumed that queries (operations) are terms formed by strings of operation symbols and variables, and assertions are formulas, i.e., strings composed from terms and logical operators.

This framework is typical for string-based algebra and logic familiar to the community, but it has a few inherited

drawbacks for generic MMT applications. First of all, string-based terms and formulas are inadequate for expressing queries and constraints over graph-based models like ER or UML diagrams. In addition, it is unclear how to define composition of mappings in this setting. Not surprisingly, in all concrete applications and implementations of this framework, it is specialized for working with relational models; see Fagin, Kolaitis, Miller, and Popa (2003); Madhavan and Halevy (2003); and Velegrakis, Miller, and Popa (2003).

A more general framework is proposed in Bernstein et al. (2000). The main idea is to reify mappings as models and to attach correspondence assertions to elements of these model mappings (see Figure 2 in *GenMMT*). In this way model mappings become spans (whose *legs*, i.e., projection arrows, are model morphisms), and their composition is defined as composition of spans. Though semantically clear, span composition is much more complicated than composition of arrows, and the problem of composing expressions still persists. A common drawback of both approaches is that queries appear as something foreign to the models and need a special interface (Problem 3 in Table 2 in *GenMMT*).

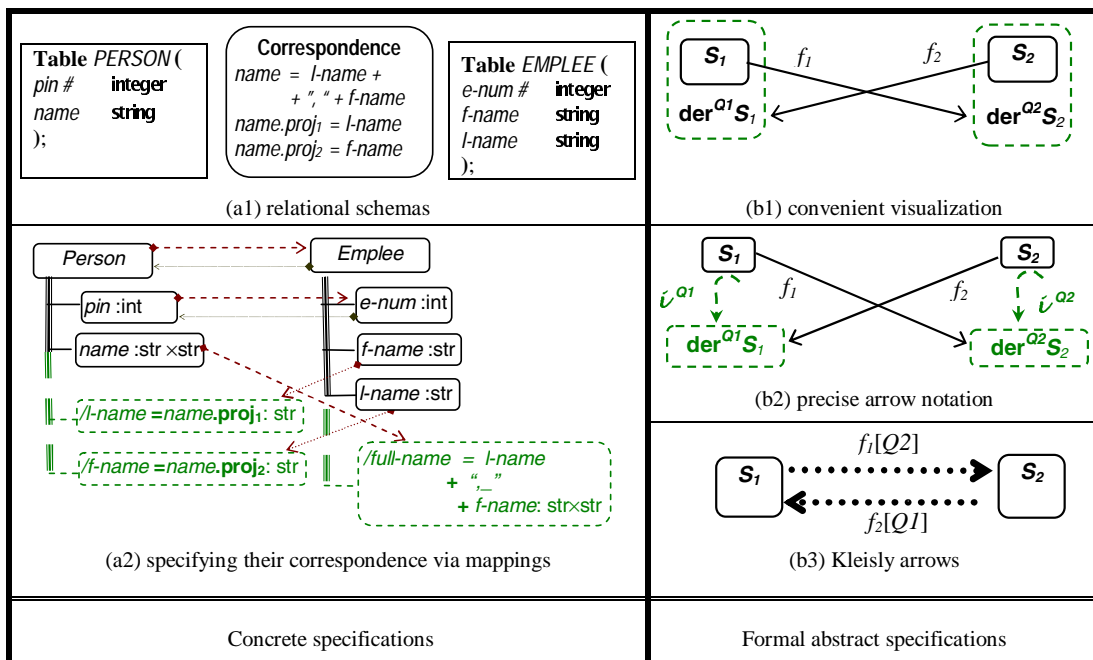
All these problems are eliminated as soon as we consider the issue in the generic framework of *categorical algebra* developed in CT in the 60th-70th (see Manes, 1976, for a basic account). The key observation is that operations/queries against the model can be denoted by adding new derived items to the model and labeling them

with the operation's name. Then correspondence between models is specified by mappings between models augmented with derived (computable) items. A simple example of how it works is presented in the left column of Figure 1. In the right column, Diagrams (b1), (b2) present an immediate abstraction of this description: Models (schemas) $der^{Q1}S_1$, $der^{Q2}S_2$ are augmentations of the original schemas with derived items produced by sets of queries $Q1, Q2$, respectively, and arrows i^{Q1}, i^{Q2} in Diagram (b2) denote inclusions of schemas.

An essential feature of the specifications in Figure 1 is that the arrows are *functional* mappings that send each item from the source to a single item in the target. We do not need to reify mappings by new intermediate models nor do we need to consider many-to-many relationships between models. Although reification of model correspondence by intermediate models (spans) is sometimes necessary, many MMT tasks can be specified with pure functional mappings (morphisms). From now on, we will use the term model mapping in the sense of functional mapping between models.

In our abstraction efforts in the right column, we can go even further and consider query expressions attributed to arrows rather than to models, as it is shown by Diagram (b3). In this way we come close to the notation with which we began our discussion, but note that in our case a figurative arrow $S \rightsquigarrow T$ is just an abbreviation of pair (Q, f) , with Q a set of queries to the target schema and $f: S \rightarrow der^{QT}$ a (functional) mapping of the source schema

Figure 1. Specifying correspondences between models via Kleisly morphisms



to the Q -augmentation of the target one. In CT, such arrows were called *Kleisly morphisms* after H. Kleisly, one of the pioneers of categorical algebra, and we will borrow this name from CT. By purely typographical reasons, Kleisly arrows in our figures are denoted by dotted-bold rather than zigzag arrows.

Kleisly Operation

A key feature of Kleisly machinery is that mappings between models can be naturally extended to mappings between augmented models. Having a mapping $f: S \rightarrow T$ and a set of queries Q over schema S , we expand $f: S \rightarrow T$ to a mapping $f^Q: \mathbf{der}^Q S \rightarrow \mathbf{der}^Q T$ by mapping each derived item $y = \mathbf{q}(x_1, \dots, x_n)$ in $\mathbf{der}^Q S$ (\mathbf{q} is a query from Q and x_i are its arguments) to an item $\mathbf{q}(x_1 f, \dots, x_n f)$ in $\mathbf{der}^Q T$, where $x.f$ denotes application of mapping f to x , that is, the f -image of x . In other words, we *homomorphically* map derived items in the source model to *similar* derived items in the target model. Further, we will call the procedure of extending a mapping to a homomorphism the *Kleisly operation*.

The composition of Kleisly morphisms $(Q_1, f_1): S \rightsquigarrow T$ and $(Q_2, f_2): T \rightsquigarrow U$ is easily defined as the ordinary functional composition

$$S \xrightarrow{f_1} \mathbf{der}^{Q_1} T \xrightarrow{f_2^{Q_1}} \mathbf{der}^{Q_1} \mathbf{der}^{Q_2} U = \mathbf{der}^{Q_1(Q_2)} U$$

where $Q_1(Q_2)$ denotes the set of query expressions obtained by substituting queries from Q_2 into queries from Q_1 .² The problem of expression manipulation engine stated in Bernstein (2003) is now resolved: The job is done by the Kleisly operation and functional mapping compositions.

Kleisly Arrows and Schema Equivalence

The Kleisly approach immediately reveals the role of derived items and queries in the issue of schema equivalence. Consider the example in Figure 1. Intuitively, it is clear that the two schemas present the same information; the only difference between them is in the choice of basic data from which other data is derived. This intuition can be formally explicated by building two mutually inverse mappings between schemas $\mathbf{der}^{Q_1} S_1$ and $\mathbf{der}^{Q_2} S_2$. The mappings are built by consecutive application of Kleisly operations to the initial configuration of schemas and mappings and state a homomorphic (with respect to the operations involved) bijection between the schemas $\mathbf{der}^{Q_1} S_1$ and $\mathbf{der}^{Q_2} S_2$. Thus, schemas $\mathbf{der}^{Q_1} S_1$ and $\mathbf{der}^{Q_2} S_2$ can be proven to be algebraically isomorphic with respect to some query language embodied into the notion of **der**-operator. In such cases we will say that schemas S_1 and S_2 are **der**-equivalent.

We will see in the next section that the notion of **der**-

equivalence is essential for schema integration: In fact, the result of schema merge is defined up to **der**-equivalence. Disregarding this peculiarity can lead to misunderstanding of the integration procedure. For example, in Buneman, Davidson, and Kosky (1992), they come to a strange conclusion that the merge operation is non-associative: In examples they consider, $(S_1 + S_2) + S_3 \neq S_1 + (S_2 + S_3)$ (“+” denotes the merge operation). The examples are right but the conclusion is not: It can be shown that actually their models $(S_1 + S_2) + S_3$ and $S_1 + (S_2 + S_3)$ are **der**^Q-equivalent for a set Q of few simple queries.

MODEL MERGE CATEGORICALLY: A HOMOGENEOUS MMt SCENARIO VIA ARROWS AND DIAGRAM OPERATIONS

Representing Models by Sketches

A vast body of work in schema integration was devoted to structural conflicts between schemas, when the same piece of reality is modeled differently in different schemas. Consider, for example, a situation when the same information is specified by an XML document and a UML diagram. Managing this diversity within syntactical frames is really hard if at all possible, and a shift to semantics is a must. The sketch format is a device for specifying semantics of a data schema in universal terms of sets and mappings irrespectively of the schema’s syntax (Diskin & Kadish, 2003). The sketch presentation of schemas provides homogeneity of their semantics and all types of structural conflicts are reduced to only one: Data considered basic in one schema are derived data in another schema (Cadish & Diskin, 1996).

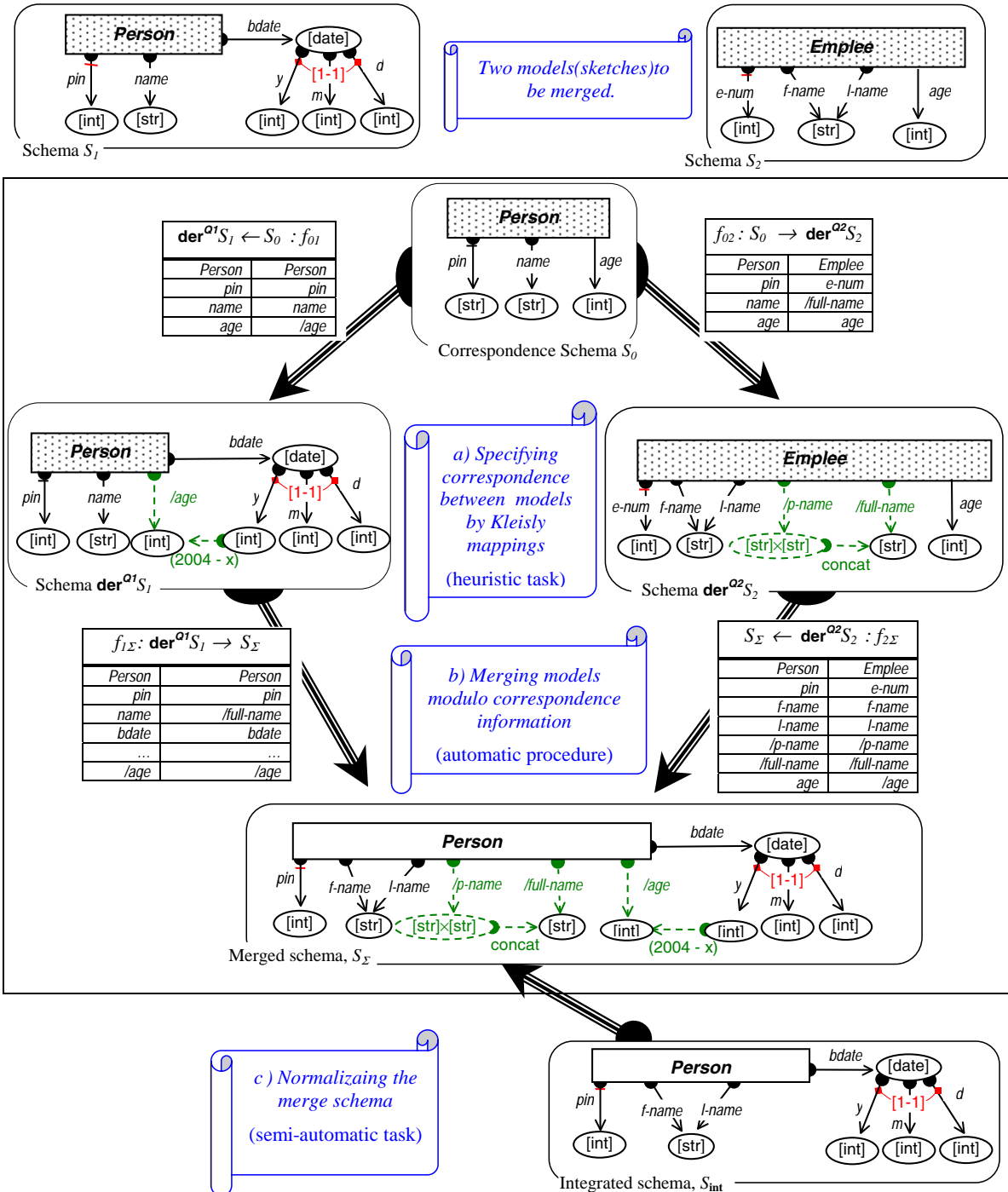
On the other hand, in many data models, representation of schemas by sketches is quite natural and easy. For example, let us consider the relational model. Semantically, rows of a relational table represent real-world objects while columns are mappings from the class of objects (rows) to value domains. From this viewpoint, it is reasonable to present a relational schema as a directed graph, whose nodes are object classes (table names) and value domains (SQL types), and arrows (columns) go from object classes to either value domains or other classes (foreign keys). For example, relational tables from Figure 2 in the previous article, GenMMt, are transformed into graphs in the upper part of Figure 2 below.

These graphs carry some additional structure visualized by markers hung on some nodes and arrows. The oval shape of a node is a visualization of declaring the node to be a value domain with predefined semantics; the latter is referred by the marker hung on the node. The bold

tail of an arrow declares the arrow to be a totally defined mapping. Marker [1-1] hung on an arrow multi-span diagram (see schema S_1) means one-one correspondence between elements of the span's source ([bdate]-values) and tuples (triples [y,m,d]) of elements taken from the targets—one element for each leg of the span. A short bar

on an arrow's tail is the same [1-1] marker hung on a span consisting of a single arrow. A graph endowed with diagram predicate declarations (like just mentioned) is called a *sketch*. Thus, relational schemas are presented by sketches.

Figure 2. Model merge concretely (in the sketch representation of models)



Merging Sketches

We specify the correspondence between schemas by Kleisly mappings between sketches. We first augment the original sketches with necessary derived items; they are shown on Figure 2 by dashed gray (green on a color display) lines.³ Then we specify overlapping between schemas by introducing a new schema (sketch) S_0 and specifying mappings from S_0 into the augmented sketches (the second row of the figure).

The next step is entirely automatic. The augmented sketches are merged modulo the correspondence specified by the span S_0 . Roughly, operation **merge** first forms a disjoint union of the sketches and then glues together items specified as equal by the span S_0 . The result is a co-span: a sketch S_Σ together with two mappings into it,

$$f_{1\Sigma}: S_1 \rightarrow S_\Sigma \text{ and } f_{2\Sigma}: S_2 \rightarrow S_\Sigma$$

The merge sketch inevitably contains redundant derived items. To finish integration, it should be *normalized*, that is, rearranged to a form with a minimal redundancy. In other words, we need to find a subsketch $S_{\text{int}} \subset S_\Sigma$ to be **der**-equivalent to S_Σ , i.e., such that $S_\Sigma \subset \mathbf{der}^{Q_{\text{int}}} S_{\text{int}}$ for some set Q_{int} of operations (queries) over S_{int} . In the present case it is fairly easy: just take the basic (black) subsketch of S_Σ . In more complex situations, finding a suitable S_{int} can be not so trivial. Some general theory of normalizing sketches would be helpful and is still waiting for its development. Clearly, normalization can be automated but, in general, a human intervention is necessary.

Towards Genericness Across Metamodels

Is the pattern of sketch merge applicable to other sorts of models? In other words, is it possible to define the merge operation in a generic way irrespectively of models' internal structure? Clearly, the key to the problem should be in semantic reformulation of merge, independent of syntactical peculiarities of the models.

Thinking semantically, given two schemas (models) S_1, S_2 , we should define their merge as the least schema containing all the information specified by S_1 and S_2 . That is, the merged schema should be rich enough (but not more than necessary) so that by making appropriate queries to it we could recover data specified by schemas S_1 and S_2 and data that can be derived from them as well.

In other words, the notion of “all the information” specified by schemas assumes, first of all, that some query language (set of operations over data) QL is given. Then any schema S can be extended to a (huge) schema $\mathbf{der}^{\text{QL}}S$ specifying all data derivable from data specified by S with

all possible queries against S . Now the merge of S_1, S_2 is defined to be a schema S_Σ such that $\mathbf{der}^{\text{QL}}S_\Sigma$ is the least schema to which schemas $\mathbf{der}^{\text{QL}}S_1$ and $\mathbf{der}^{\text{QL}}S_2$ could be mapped:

$$\mathbf{der}^{\text{QL}}S_1 \xrightarrow{f_{1\Sigma}} \mathbf{der}^{\text{QL}}S_\Sigma \xleftarrow{f_{2\Sigma}} \mathbf{der}^{\text{QL}}S_2$$

Of course, in concrete applications it is sufficient to deal with finite fragments of full **der**^{QL}-completions, that is, schemas $\mathbf{der}^Q S$ for some finite set Q of queries against S .

If the schemas overlap in information they specify, we need to refine this description. Overlapping is specified by some correspondence schema S_0 encompassing the common items, and two mappings to show where these common items are placed in the schemas S_1, S_2 , that is, by a span:

$$\mathbf{der}^{\text{QL}}S_1 \xleftarrow{f_{01}} S_0 \xrightarrow{f_{02}} \mathbf{der}^{\text{QL}}S_2$$

over the schemas to be related. Merge of S_1, S_2 modulo this correspondence is a schema S_Σ as above and, in addition, arrow compositions $f_{01} \gg f_{1\Sigma}$ and $f_{02} \gg f_{2\Sigma}$ must be equal (see Cell (b) in Table 1).

The $\mathbf{der}^{\text{QL}}S_\Sigma$ -schema's property of being “the least” can be explicated in the following (typical for CT) way. For any other schema T with mappings:

$$\mathbf{der}^{\text{QL}}S_1 \xrightarrow{f_{1T}} T \xleftarrow{f_{2T}} \mathbf{der}^{\text{QL}}S_2$$

such that compositions $f_{01} \gg f_{1T}$ and $f_{02} \gg f_{2T}$ are equal, there is a unique mapping:

$$!: \mathbf{der}^{\text{QL}}S_\Sigma \longrightarrow T$$

such that all the arrow diagrams involved are commutative. In this way we come to nothing but the definition of the join (push-out) operation well known in CT; see Table 2. In other words, it is reasonable to explicate the informal notion of schema merge (motivated by its practical applications and intuition) by the formal notion of join operation in a suitable category of schemas.

Now our procedure of merging sketches can be abstractly specified as shown in Cells (a), (b), and (c) of Table 1, where marker [join] on a square diagram refers to an abstract operation defined in Table 2. The rightmost cell presents an abbreviated specification via Kleisly arrows. Specifications in Table 1 are applicable to any category of models, not only sketches, where a Kleisly structure and the diagram operation **join** are defined.

Table 1. Model merge abstractly

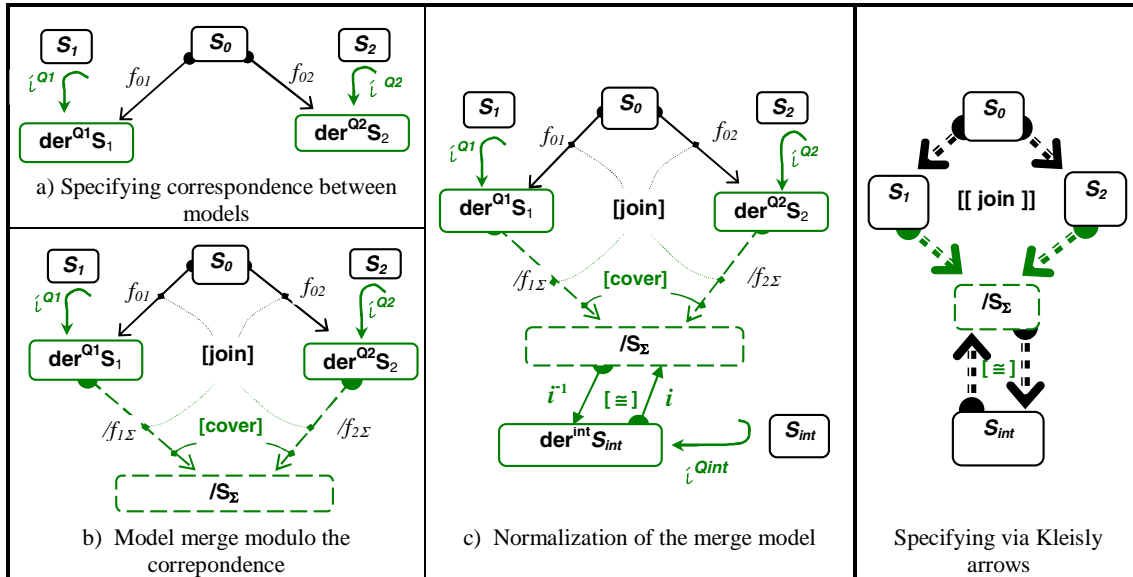


Table 2. A few basic diagram operations

| Name and denotation | Input Diagram | Output Diagram | Semantics (informally) |
|---|---------------|----------------|--|
| Composition [comp] | | | Composition of mappings |
| Image [img] | | | A sort of image (range) of a mapping: $/I$ is the smallest subobject of T (i.e., there 1-1 mapping from $/I$ into T) such that f factors through it. |
| Push-out (categorical join) [join] | | | A sort of join (merge): $/J$ is the smallest object “containing” objects T_1, T_2 w.r.t. their “common” part S . If f_2 is 1-1 mapping, then $/f_2^*$ is also 1-1 and $/J$ can be considered as T_1 augmented with the “extra”-part of T_2 w.r.t. S . |
| Pull-back (categorical meet) [meet] | | | A sort of meet (intersection): $/M$ is the largest object “contained” in objects T_1, T_2 w.r.t. their “embedding” into S . If f_2 is 1-1, then $/f_2^*$ is also 1-1 and $/M$ can be considered as the <i>coimage</i> of T_2 under f_1 , i.e., the largest subobject of T_1 whose image under f_1 is inside T_2 . |
| Difference [diff] | | | $/D$ is the smallest subobject of S , whose (categorical) union with T gives S . |

MATHEMATICS OF HOMOGENEOUS GENERIC SPECIFICATIONS

Various metadata management tasks may require various operations with models. For example, we may need to extract the common part of two submodels of a given model (intersection) or to form a submodel complementing some given submodel to the entire model. Likewise, given a model mapping $m: S \rightarrow T$, we may need to have m -image of some submodel of S or m -coimage of T 's submodel. In general, MMT may require model counterparts of many familiar operations with sets and mappings (functions).

Specifying such operations for a particular type of model (relational schemas, XML DTDs, sketches) would not be a serious problem. However, for generic MMT we need to specify these operations in a metamodel independent way, that is, independently of syntactic peculiarities of models whatever they might be. At first glance, the problem looks unapproachable and the few attempts published in the literature tried to avoid it by using “cumulative generalization” rather than abstraction. The idea was to catalogue the constructs used in all the dialects of a modeling language and try to extract a smaller generating subset from them. Examples are “the most general” ER format by Atzeni and Torlone (1996) or “the most general” metamodel for OO modeling defined in the Object Management Group's (2002) Meta-Object Facility specification. Clearly, this sort of genericness is very restrictive and for model translation does not work at all.

A proper approach to the problem was found in category theory. The latter demonstrated that familiar operations with sets and mappings can be abstractly reformulated for objects of arbitrary nature in terms of arrows (mappings), without any references to objects' internal structure. A few examples of such a reformulation are presented in Table 2. Basically, they follow the pattern we have used in the previous section for abstract reformulation of model merge.

One can note that an essential part of these formulations is a requirement of being the smallest/largest object satisfying certain properties. Such a requirement is called a *universal property* and can be also formulated via arrows similarly to how we described the minimality of model merge in the previous section. A surprising result discovered in category theory is that there is a small basic set of universal operations from which an extremely rich set of other diagram operations can be generated by composition. Categories (universes of objects and mappings) possessing this set of operations are called *toposes*, and they are fairly remarkable. Namely, every constructive operation over sets and mappings, which one normally enjoys in a set universe, can be defined in an arbitrary topos as well.⁴

Thus, given a collection \mathcal{M} of similar models and mappings, if we have implemented a few diagram operations constituting \mathcal{M} as a topos, then by composing these operations we can also perform any constructive operation with models and mappings. Of course, this nice picture needs some reservations and details, and we will return to it in the last section of *Math-II*. First, however, we need to figure out whether this framework is also suitable for heterogeneous MMT—so far, our models were supposed to be similar. A priori, it is not clear how to manage a key MMT operation of data and schema translation in a generic way. This is the goal of the next article.

REFERENCES

- Atzeni, P., & Torlone, R. (1996). Management of multiple models in an extensible database design tool. In *Lecture notes in computer science: Vol. 1057. International Conference on Extending Database Technology, EDBT'96* (pp. 79-95). Springer Verlag.
- Bernstein, P. (2003). Applying model management to classical metadata problems. In *International Conference on Innovative Database Research, CIDR'2003* (pp. 209-220).
- Bernstein, P., Halevy, A., & Pottinger, R. (2000). A vision for management of complex models. *SIGMOD Record*, 29(4), 55-63.
- Buneman, P., Davidson, S., & Kosky, A. (1992). Theoretical aspects of schema merging. In *Lecture notes in computer science: Vol. 580. International Conference on Extending Database Technology, EDBT'92*. Springer.
- Cadish, B., & Diskin, Z. (1996). Heterogeneous view integration via sketches and equations. In *Lecture notes in artificial intelligence: Vol. 1079. Foundations of Intelligent Systems, Ninth International Symposium, ISMIS'96* (pp. 603-612). Springer Verlag.
- Diskin, Z., & Kadish, B. (1997). A graphical yet formalized framework for specifying view systems. In *Advances in Databases and Information Systems, ADBIS'97: Vol. 1. Regular papers* (pp. 123-132). St. Petersburg, Russia: Nevsky Dialect.
- Fagin, R., Kolaitis, P., Miller, R., & Popa, L. (2003). Data exchange: Semantics and query answering. In *International Conference on Database Theory, ICDT'2003* (pp. 207-224).
- Lenzerini, M. (2002). Data integration: A theoretical perspective. In *ACM Symposium on Principles of Database Systems, PODS'2002* (pp. 233-246).

Manes, E. (1976). *Graduate texts in mathematics: Vol. 26. Algebraic theories*. Springer-Verlag.

Object Management Group. (2002). *Meta-Object Facility (MOF) specification, Version 1.4*. Retrieved from <http://www.omg.org/technology/documents/formal/mof.htm>

Velegrakis, Y., Miller, R., & Popa, L. (2003). Mapping adaptation under evolving schemas. In *International Conference on Very Large Databases, VLDB'2003*.

KEY TERMS

Category Theory (CT): A branch of modern algebra, providing a language and machinery for defining and manipulating mathematical structures in an abstract and generic way. The method of CT is to present the universe of discourse as a collection of objects (nodes) and morphisms (arrows) between them. The latter can be composed and so the universe is presented as a *category*: a directed graph with composable arrows. For example, models of a given sort (metamodel) and mappings between them form a category in a very natural way. If one wants to work with a heterogeneous universe of models (that is, models of different metamodels), one should use *fibrations* described in *Math-II*.

The philosophy of CT is that everything one wants to say about objects, one must say in terms of arrows between objects. Correspondingly, an object's structure is a structure over its arrow interface. This way of defining structures and manipulations with them is often called *arrow thinking*.

Diagram Operation: An operation over objects and morphisms that takes a certain configuration of them as an input, *input diagram*, and augments it with a few new—derived—objects and morphisms matching the shape of the *output diagram*. Table 2 presents a few examples: Derived items are shown with dashed gray (green on a color display) lines and their names are prefixed by slash (UML's notation).

Diagram Predicate: A property of a certain configuration, *diagram*, of objects and morphisms.

Kleisly Morphism: A basic construct of categorical algebra. In the context of MMT, a Kleisly morphism from model S to model T is a mapping of the source model's items to derived or basic items of the target model. In other words, a Kleisly morphism assumes (i) a set of queries Q to the target model and (ii) a mapping (function) $m: S \rightarrow \mathbf{der}^Q T$ into the augmentation of the target model with derived items corresponding to the queries from Q .

Sketch: A basic construct of categorical logic. Roughly, it is a directed graph in which some diagrams are marked with predicate symbols taken from a predefined signature. A sketch morphism is a mapping of the underlying graphs compatible with marked diagrams. A fact of fundamental importance for generic MMT is that any metamodel (relational, XML Schema, a specific sort of ER or UML diagram) can be specified by a sketch. Then the corresponding universe of models (relational schemas, XML DTDs, ER diagrams) can be presented as the category of instances of the corresponding sketch. It allows applying many CT ideas and results to MMT.

ENDNOTES

- 1 Partially supported by Grant 05.1526 from the Latvian Council of Science.
- 2 To make this description precise, we need a formal and generic notion of query, that is, an operation over data specified by a schema. Such a notion was defined in Diskin and Kadish (1997).
- 3 Derived items have to be added to sketches only within a strict discipline of applying diagram operations. For example, in sketch $\mathbf{der}^{Q_2} S_2$, the operation **pair** takes two arrows $fname$, $lname$ as input and produces an arrow $pname$ (paired-name). Then composition of arrows $pname$ and $concat$ results in an arrow $fullname$. For sketch $\mathbf{der}^{Q_1} S_1$, arrow age is defined as composition of arrows $bdate \gg y \gg (2004 - *)$. To distinguish between basic and derived items, the latter are shown in dashed gray (green on a color display) lines to suggest that extent of these items is computable and does not need to be stored. In addition, names of derived items are prefixed with slash, as suggested by UML.
- 4 However, in a topos these operations have unusual properties. For example, if T is a subobject of object S , and $S-T$ is (a sort of) their difference (see Table 2), intersection of T and $S-T$ is not necessarily empty. And this is just what we have in MMT with graph representation of models: Subgraphs T and $S-T$ do not have common arrows but can have common nodes. The logic/algebra of toposes can be precisely specified and turned out to be an algebraic version of the so-called intuitionistic higher-order predicate logic rather than classical two-valued first-order logic.

Mathematics of Generic Specifications for Model Management, II

Zinovy Diskin¹

SWD Factory, Latvia and Universal Information Technology Consulting, USA

This article (further referred to as Math-II), and the previous one (further referred to as Math-I, see p. 351), form a mathematical companion to Generic Model Management (further referred to as GenMMt, see p. 258). While Math-I is dealing with homogeneous MMt, the goal of Math-II is to develop machinery for heterogeneous MMt, where models are not assumed to be similar.

INTRODUCTION

The main issue of heterogeneous MMt is data and model transformation from one format (metamodel) into another. A general setting for the task is as follows: Data (instance) D_1 over a schema S_1 in metamodel M_1 needs to be transformed into data D_2 over a schema S_2 in metamodel M_2 . So far, a generic format for the task has not been described in the literature. However, the problem can be solved in the arrow framework: Table 1 presents the entire operation as a sequence of standard diagram operations with data, models, metamodels, and mappings between them. The diagrams in the table look simple, but they are actually abbreviations of more complex constructs. To understand these abbreviations, we will first consider a simple example of the task and then abstract it to a generic specification.

GETTING STARTED: AN EXAMPLE

Suppose we need to translate data stored in the graphic sketch format into relational tables. This task is reasonable only if the relational schema elements—tables, columns, and primary and foreign keys—are somehow implicitly present in the sketch format elements—nodes, arrows, and diagram predicates. To perform the task, we need to make this presence explicit and build a mapping from the metaschema of relational schemas into the metaschema of sketches.

For this purpose, we need to specify the metaschemas in some common format. For instance, we could specify them as relational schemas (then data schemas will be treated as relations) or sketches (then data schemas will be sketch instances, i.e., special graphs). In our example

we chose sketches; this format is universal and convenient for presentation. Thus, we need to build a sketch M_{rel} specifying the relational metaschema; a sketch M_{ske} specifying the metaschema of sketches; and a sketch mapping (Kleisly arrow) $m: M_{rel} \rightarrow \mathbf{der}^Q M_{ske}$ interpreting relational constructs by either basic or derived constructs of the sketch format.

To simplify presentation, we will consider a simple version of relational schemas, where the primary key consists of only one column. This version is specified by sketch M_{rel} in the lower right cell of Figure 1 (for a while, do not pay attention to the right parts of nodes' and arrows' names following after colons). An instance of this sketch as a data schema is a graph S labeled by M_{rel} 's items in such a way that the labels set a graph mappings $s: S \rightarrow M_{rel}$. An example is presented in the cell just above the M_{rel} cell. In that graph, $/S^*$, objects' names (identifiers) are prefixed with # while values are braced with “”; names of arrows are omitted. Nodes' and arrows' labels follow their names after the colons and are violet on a color display. Such labeled graphs are, in fact, graph mappings as shown by the bold dotted arrow near the left border of the cell. The source of such a mapping is the graph itself while the labels determine the mapping into the graph pointed by the bold dotted arrow. It is easy to see that mapping a graph into the relational metamodel sketch M_{rel} makes it a relational schema. For example, the labeled graph $/S^*$ in Figure 1 is nothing but the relational schema specified in the cell on the right. These two cells present the same metadata.

A sketch specifying an essentially simplified sketch metamodel, M_{ske} , is shown in the left lower cell of the figure (do not consider dashed items for a moment). This metamodel says that a sketch consists of *Node* objects and *Arrow* objects, and each *Arrow* is assigned with two *Nodes* called its *source* and *target*. The class of nodes is partitioned into *objectClasses* and *valueDomains*; the former have *names* (that are *strings*, i.e., elements of the domain [str]), while the latter have *SQL types*. In addition, each *objectClass* node is assigned an arrow called the *key* to the class, and the subgraph consisting of these *key* arrows must be acyclic (constraint **KGA: Key (sub)Graph is Acyclic**).² An instance of this sketch is a labeled graph, that is, a graph mapping $\sigma: S \rightarrow M_{ske}$, like that one shown in the cell just above the metaschema M_{ske} . More compact

presentation of this instance is shown on the left.

For interpreting items of the relational metaschema, sketch M_{ske} must be augmented with the corresponding derived items. They are shown by dashed gray (green on a color display) lines and defined as specified by Figure 1A; their names are prefixed with slash, as suggested by UML. Now we can explicate correspondence between sketch and relational schemas by building an interpretation mapping $m: M_{\text{rel}} \rightarrow \mathbf{der}^Q M_{\text{ske}}$. This mapping is specified in Figure 1 by labeling the M_{rel} 's items by M_{ske} 's items; labels stand after colon in M_{rel} -items' names and are shown in violet on a color display.

DATA TRANSLATION IS DETERMINED BY METAMODEL INTERPRETATION

We will see how metamodel interpretation governs the entire transformation procedure. To illustrate the idea, we apply the general procedure to a very simple piece of data shown in the left-most upper cell of Figure 1.

The first step is to present data as a graph mapping $\delta: D \rightarrow S$ from the data graph to the schema graph, and the schema (data on the metalevel) as a graph mapping $\sigma: S \rightarrow M$ into the metaschema graph. These mappings, specified via labeling (as described above), are shown in the left column half of Figure 1 (by the abuse of notation, we use labels of schema arrows as their “names” for labeling the data-graph arrows). Identifiers for the four data links in the leftmost top cell are denoted in graph D by $\#P_i$, $i=1, \dots, 4$).

Definitions of derived items augmenting the metaschema are query specifications for its instances—schemas. Executing these queries augments the schema with derived items shown in the dashed lines, which, in their turn, are query specifications for data and propagate to augmenting data. In this way we obtain a chain of extended mappings $\overline{D} \xrightarrow{\overline{\delta}} \overline{S} \xrightarrow{\overline{\sigma}} \overline{M} = \overline{M_{\text{ske}}}$, where overlined capital letters denote augmented graphs. To simplify notation, further in the article we will always interpret arrows by Kleisly mappings and omit bars over objects' and arrows' names. In other words, we will work in a Kleisly category over graphs (determined by a choice of legitimate operations/queries with data; see *Math-I*).

The next step is to select that part of the schema sketch S , which is mapped by σ into the range of the interpretation m , and rearrange its labeling to sketch M_{rel} . This is done by applying to the span $S \xrightarrow{\sigma} M_{\text{ske}} \xleftarrow{m} M_{\text{rel}}$ the diagram operation of **meet** explained in Table 2 of *Math-I*. The result is a new graph $/S^*$ together with two mappings:

vertical, $/\sigma^*: /S^* \rightarrow M_{\text{rel}}$, and horizontal, $S \xleftarrow{/m^*} /S^*$.³

The vertical mapping is shown in the right middle cell of Figure 1; it makes $/S^*$ a relational schema. The horizontal mapping (informally described by the visual graphic similarity between $/S^*$ and S) relates $/S^*$ with the original schema sketch and shows how the labels are changed.

Finally, we select that part of the data graph D , which is mapped by δ into the range of the schema mapping $/m^*$, and rearrange its labeling to graph $/D^*$. This is again done by applying the **meet**-operation and produces a new graph $/D^*$ together with two mappings:

vertical, $/\delta^*: /D^* \rightarrow /S^*$, and horizontal, $D \xleftarrow{/m^{**}} /D^*$.

The former mapping makes data graph $/D^*$ an instance over schema $/S^*$, and the latter relates it to the original data graph.

Properties of the interpretation m determine properties of the entire procedure. Since in our example m is injective (one-one) and the **meet**-operation preserves injectivity, objects $/S^*$ and $/D^*$ are, in fact, sub-objects of their sketch-format counterparts. In fact, we did nothing but select these sub-objects and rearrange their labeling. Thus, for one-one mapping m , **meet** works with labels of data items rather than with data itself.

Another special property of our example is that the schema sketch S is so simple that the range of mapping σ comes to be inside of the range of mapping m , i.e., inside of the image of M_{rel} under m . Hence, no data item was lost during the transformation. In general, the full sketch format is richer than relational, and some part of the full sketch metamodel goes beyond the range of m . If a piece of data, D , has a schema whose image under mapping σ goes beyond the range of m , then some data items from D will be lost during the transformation.

Considering this simple example in abstract terms leads to a generic pattern of data and schema transformation. It is described in the next section.

DATA TRANSFORMATION VIA DIAGRAM CHASING

Table 1 presents an abstract generic specification of data transformation in the general setting described at the beginning of the article. The input information for the entire procedure is formed by (1) a data instance D_1 over a source schema S_1 and (2) two Kleisly morphisms: metaschema interpretation $m_{12}: M_2 \rightsquigarrow M_1$ and schema mapping (view) $s_{12}: S_2 \rightsquigarrow /S_1^*$ of the target schema into the M_2 translation of the source schema. All the rest is done automatically by consecutive application of the two ge-



Table 1. Generic format of data transformation

| Heterogeneous MMt | | Homogeneous MMt | |
|---|--|--|--|
| Step 1 | Step 2 | Step 3 | Step 4 |
| <p>Specifying 1st half of input data for the operation: metaschema interpretation (Kleisly arrow) m_{12}</p> | <p>Translating schema S_1 into schema S_1^* in metamodel M_2 and then rearranging the data D_1 to data $/D_1^*$ over schema $/S_1^*$</p> | <p>Entering 2nd half of input data for the operation: schema S_2 in metamodel M_2 together with a view (Kleisly arrow) s_{12} to schema $/S_1^*$</p> | <p>Rearranging data $/D_1^*$ into data $/D_1^{**}$ over schema S_2.</p> |
| <p>Quasi-homogeneous MMt: Overall specification of Steps 1..4 via Grothendieck arrows and operations with them</p> | | | |

neric diagram operations: Kleisly expansion for vertical arrows (see section “Kleisly Operation” in *Math-I*) and **meet** for the left-lower arrow spans (Table 2 in *Math-I*).

In more detail, after we have translated data and schema over metamodel M_1 into metamodel M_2 (see Steps 1 and 2 in Table 1), we can safely specify the target schema S_2 as a view to the schema $/S_1^*$, that is, as a (homogeneous) Kleisly mapping $s_{12}: S_2 \rightsquigarrow /S_1^*$ (Step 3 in Table 1). Computing data D_2 is now nothing but computing a (materialized) view of data D_1 specified by schema S_2 . This computation is performed by applying the diagram operation **meet** once again (Step 4).

Note, the metaschema interpretation m_{12} sets the base of the entire procedure and in much determines its properties. Strictly speaking, we cannot even talk about schema S_2 as a view to schema S_1 until S_1 is translated to a form similar to S_2 (to S_2 metaschema’s terms). On the other hand, we can introduce a new sort of arrows between schemas, say, $f: S \rightrightarrows T$, which would be a shorthand for pairs (m, s) with $m: M_S \rightsquigarrow M_T$, a Kleisly arrow (itself a shorthand!) between metaschemas and $s: S \rightarrow /T^*$, a homogeneous schema mapping. The target of this mapping, $/T^*$, is the translation of schema T into the M_S terms according to interpretation m as described above. We will call such arrows *Grothendieck morphisms* or *mappings*, after Alexander Grothendieck, the inventor of this fundamental categorical construct.

Grothendieck mappings provide a very convenient abbreviation. With Grothendieck arrows we can specify heterogeneous MMt procedures as if we are living in a homogeneous model world. For example, the task of data transformation looks like a simple view computation; see the lower half of Table 1. For another example, heterogeneous model merge can be specified as shown in the right-most column of Table 1 in *Math-I* but with Kleisly arrows replaced by Grothendieck arrows. Implementation, however, needs de-abbreviating the Grothendieck arrows and operations with them into arrows between model representations and operations over them, as shown in the upper half of Table 1 here.

CATEGORICAL FOUNDATIONS FOR GENERIC MMt SPECIFICATIONS

Consider specifications in Table 1 of *Math-I* and Table 1 in this article once again. We have directed graphs, whose nodes denote data and schema representations, and arrows are mappings between them. The graphs are endowed with marked diagrams denoting either operations with representations and their mappings, for example, [meet] and [join], or properties of their configurations, for example, in cell (b) of Table 1 in *Math-I*, the pair of arrows $/f_{1\Sigma}, /f_{2\Sigma}$ is declared to be *covering* as specified by the marker [cover]. Another example of a diagram

Table 2. Abstract MMt in categorical light

| | Elementary Units | Repository Structure | Elementary Query | A Complete Query Language |
|------------------|------------------|-------------------------------|----------------------|--|
| Data Management | Value | Set of relations (tables) | Relational operation | Relational algebra (a version of first-order predicate calculus) |
| Model Management | Model, mapping | Sketch of models and mappings | Diagram operation | MMt doctrine/topos (a version of higher-order intuitionistic predicate calculus) |

property is commutativity of all arrow squares involved (by default). Thus, our meta-specifications are sketches in some signature of diagram operations and predicates.

This format is generic across metamodels simply because metamodels are included into specifications, either explicitly, as in the upper part of Table 1, or implicitly, when we interpret arrows by Grothendieck mappings. The format is also generic across applications: Other MMt tasks may require other diagram operations and predicates to be applied, but the very pattern remains the same. Since operations involved should inevitably include arrow composition, the sketches we discuss are finite presentations of *categories* endowed with an extra structure.

This extra structure consists of two components. The first is a Kleisly structure, \mathbf{Q} . It is formed by query operations of augmenting nodes (representations) with new items and the corresponding expansions of arrows (see section “Model Mappings as Kleisly Arrows” in *Math-I*). The second component is an algebra \mathbf{A} of diagram operations (over representations and their mappings as holistic entities); see Table 2 in *Math-I* for examples. Using the notion of Kleisly mapping, we can “hide” the structure \mathbf{Q} inside of the arrows and consider diagram operations over the Kleisly category $\mathbf{R}[\mathbf{Q}]$. Thus, the MMt universe can be considered as a pair $(\mathbf{R}[\mathbf{Q}], \mathbf{A})$ with $\mathbf{R}[\mathbf{Q}]$ a category of model representations and Kleisly mappings between them, and \mathbf{A} a diagram algebra over it.

A question of fundamental importance for gMMt is whether there exists a set of diagram operations whose compositions can generate any MMt operation of practical interest (of course, here we do not consider heuristic MMt operations like schema matching). This sort of problem was studied in Category Theory in detail, and roughly the results are as follows.

Endowing a category \mathbf{C} with a certain set of diagram operations \mathbf{A} allows interpreting logical calculi in \mathbf{C} , and the other way round, a pair (\mathbf{C}, \mathbf{A}) generates a certain logical calculus (the so-called internal language of \mathbf{C}). In this way a category endowed with an extra algebraic structure and a logical calculus turn out to be basically equivalent constructs.⁴ That is why pairs (\mathbf{C}, \mathbf{A}) are called (*logical*) *doctrines* in categorical logic.

An important aspect of the notion of doctrine is that

the component \mathbf{A} actually refers to the entire pool of diagram operations legitimate over \mathbf{C} rather than to a specific set of basic operations generating \mathbf{A} by composition. Such pools of operations closed under composition are sometimes called *clones*. Thus, the notion of doctrine refers to a clone of diagram operations rather than to a specific set of operations generating it.⁵

A fine hierarchy of doctrines and their calculus counterparts was built in Category Theory. In the top part of the hierarchy are algebraically and logically rich categories called *toposes*, whose logical calculus counterparts are higher-order predicate logics, type theories, and constructive set theories. Roughly, in a topos one can perform all constructive operations with objects which one would normally enjoy with ordinary sets; particularly, take intersections and unions of objects, form their images and co-images under given morphisms, consider graphs (binary relations) of morphisms, take transitive closures of binary relations, and much more. Surprisingly, this extremely rich pool of manipulations with objects is generated by compositions of only three basic diagram operations: *arrow composition*, *meet*, and *powerset* (building the object of all sub-objects of a given object).

The category of graphs is proven to be a topos, and hence, every constructive operation with graphs can be presented as a composition of three basic topos operations. However, the *powerset* operation is extremely non-effective and, hence, should be excluded from the clone of MMt operations \mathbf{A}_{MMt} . Thus, the latter is a proper subclone of the clone of topos operations \mathbf{A}_{Top} , $\mathbf{A}_{MMt} \subset \mathbf{A}_{Top}$. Since *powerset* is excluded from \mathbf{A}_{MMt} , we face the problem of finding an appropriate set of basic operations generating the entire clone \mathbf{A}_{MMt} . This set should include *image*, *meet*, *join* (see Table 2 in *Math-I*), some restricted version of *powerset* (discussed in Diskin & Kadish, 2003), and, perhaps, a few other important operations, which could be derived (in a topos) with the powerset operation but without it must be included into the basic set. Finding the latter is currently an open question. Nevertheless, there should be an object $(\mathbf{C}_{MMt}, \mathbf{A}_{MMt})$ in the hierarchy of doctrines—let us call it *MMt doctrine* or *MMt topos*—in which algebra \mathbf{A} consists of effective diagram operations covering all gMMt’s needs.⁶

Figure 1. Example of data translation: sketches into relational tables

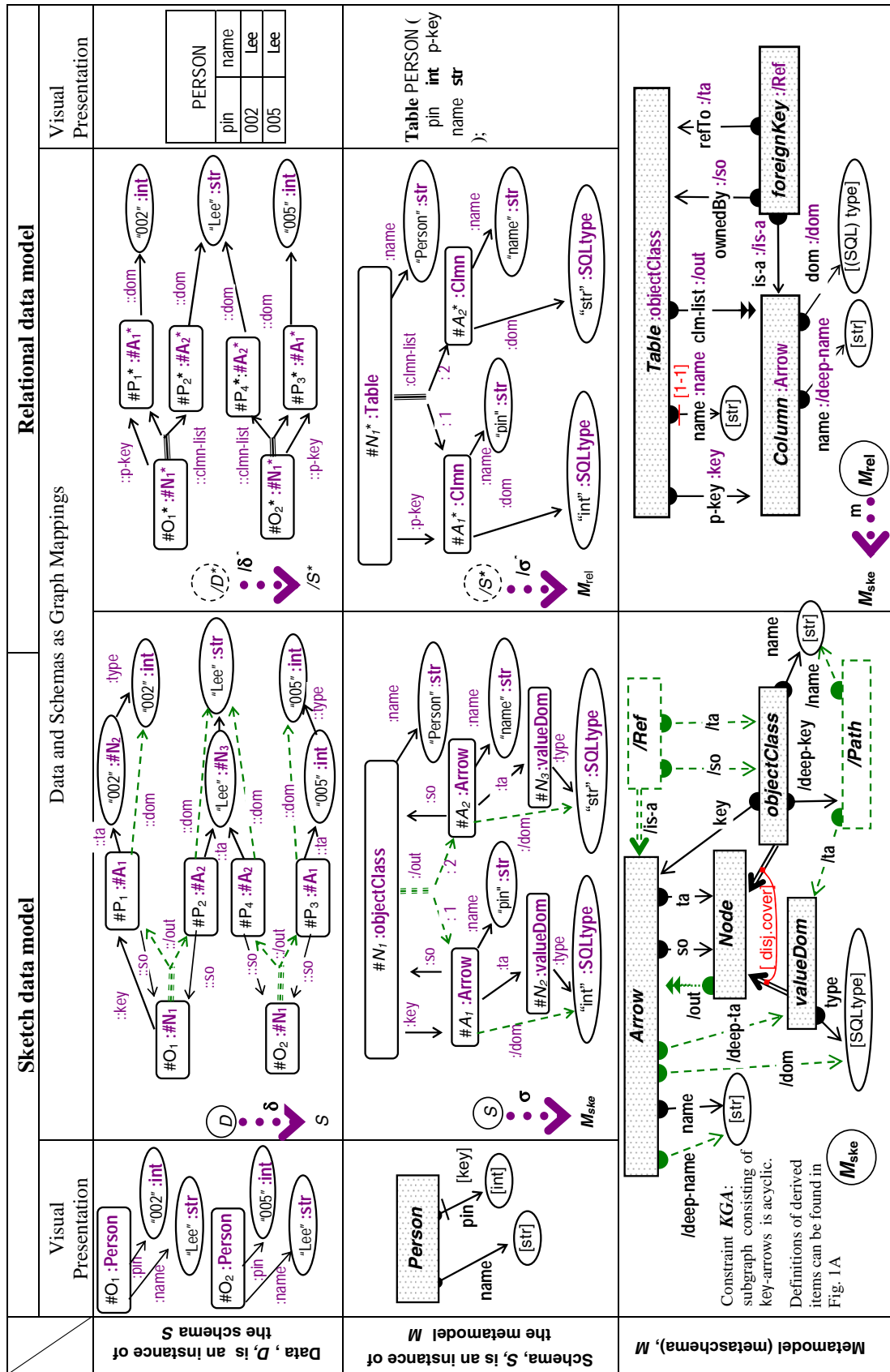


Figure 1A. Set $Q = \{Q1, \dots, Q9\}$ queries specifying derived items of sketch M_{ske} in Figure 1. An extent of item X (a node or an arrow) is denoted by $[[X]]$ (a set or a mapping resp.)

| | | |
|------|------------------|--|
| | | The intended extents of derived nodes are defined as follows: |
| (Q1) | $[[/Ref]]$ | $\stackrel{def}{=} \{A \in [[Arrow]] \mid A.[[ta]] \in [[objectClass]]\}$ |
| (Q2) | $[[/Path]]$ | $\stackrel{def}{=} \{(A_1, \dots, A_n), A_i \in [[Arrow]] : A_1.[[so]] \in [[objectClass]], A_i.[[ta]] = A_{i+1}.[[so]], A_n.[[ta]] \in [[valueDom]]\}$ |
| | | The extents of derived arrows are defined as follows: for any $A \in [[Arrow]], P \in [[Path]], N \in [[Node]], C \in [[objectClass]]$ |
| (Q3) | $P.[[ta]]$ | $\stackrel{def}{=} A_n.[[ta]]$ |
| (Q4) | $P.[/name]$ | $\stackrel{def}{=} "A_1.[[name]]_ " A_2.[[name]]_ \dots _ " A_n.[[name]]\{"}$ |
| (Q5) | $N.[/out]$ | $\stackrel{def}{=} \text{the list of all } (A_1, \dots, A_k) \text{ with } A_i.[[so]] = N,$ |
| (Q6) | $C.[/deep-key]$ | $\stackrel{def}{=} \text{CASE } \{ \begin{array}{l} (C.[[key]]) \text{ if } C.[[key]].[[ta]] \in [[valueDom]]; \\ (C.[[key]] \mathbf{append} C.[[key]].[[ta]].[/deep_key]) \text{ if } C.[[key]].[[ta]] \in [[objectClass]] \end{array} \},^*)$ |
| (Q7) | $A.[/deep-ta]$ | $\stackrel{def}{=} \text{CASE } \{ \begin{array}{l} A.[[ta]] \text{ if } A.[[ta]] \in [[valueDom]]; \\ A.[[ta]].[/deep-key].[/ta] \text{ if } A.[[ta]] \in [[objectClass]] \end{array} \},$ |
| (Q8) | $A.[/dom]$ | $\stackrel{def}{=} A.[/deep-ta].[[type]]$ |
| (Q9) | $A.[/deep-name]$ | $\stackrel{def}{=} \text{CASE } \{ \begin{array}{l} A.[[name]] \text{ if } A.[[ta]] \in [[valueDom]]; \\ "A.[[name]]_ " A.[[ta]].[/deep-key].[/name]\{" \text{ if } A.[[ta]] \in [[objectClass]] \end{array} \}$ |
| | | <small>*) It's a recursive definition. It provides a well-defined result owing to the constraint KGA and finiteness of the graph</small> |

Now we can fill in our Table 1 of abstract MMT's goals from *GenMMt* as shown in Table 2.

CONCLUSION

Software engineering (like any other engineering field) is governed mainly by technological and economical reasons near the ground, rather than criteria of intellectual beauty and elegance in the sky. Nevertheless, it appears that for MMT, abstract compact specifications of high generality are "a matter of life and death rather than disposable luxury".⁷ Thus, it is nothing surprising in that the 30-year history of managing metadata has led to the "abstract-intensive" discipline of *generic MMT*.

A similar need in genericness emerged in mathematics in the 1940s, after many years of playing with mathematical structures, their mappings and transformations, and Category Theory was born. It offered a language and machinery for generic structure engineering, and one might expect that it could be useful for gMMt as well. Nevertheless, even with these considerations in mind, it still seems mysterious that constructs and ideas developed in an entirely abstract mathematical discipline appear as if specially designed to manage practical problems. On the other hand, many pieces of categorical

methods and constructions have already appeared in MMT literature and implemented in software independently of Category Theory (CT). Though they are often implicit and partially corrupted and not mathematically clean, it would not be an overstatement to say that today software engineers and researchers are rediscovering categorical specification patterns in an industry-adaptable form.

Having precise mathematical counterparts of generic MMT specifications would help MMT greatly. Moreover, what gMMt could borrow from CT does not amount to a few constructs and methods. In more than 50 years of work in pure mathematics and applications, CT has developed a special intuition and consistent methodology for specifying, reasoning, and operating complex structures in a generic way, which is often called *arrow* thinking. What gMMt should borrow from CT first of all is the arrow style of thinking, after which categorical methods and machineries could be properly adapted and applied to MMT problems.⁸

REFERENCES

Barr, M., & Wells, C. (1995). *Category theory for computing science*. Prentice Hall International.

Diskin, Z. (1998). *Representing objects by values: Towards semantic foundations for object-oriented relational database systems*. (Tech. Rep. FIS-LDBD-9802). Frame Inform Systems, Riga, Latvia.

Diskin, Z. (2001). On modeling, mathematics, category theory and RM-ODP. In H. Kilov (Ed.), *WOODPECKER'2001: Workshop on Open Distributed Processing: Enterprise, computation, knowledge, engineering and realization* (pp. 38-54). Lisbon, Portugal: ICEIS Press.

Diskin, Z., & Kadish, B. (2003). Variable set semantics for keyed generalized sketches: Formal semantics for object identity and abstract syntax for conceptual modeling. *Data & Knowledge Engineering*, 47, 1-59.

Freyd, P., & Scedrov, A. (1990). *Categories, Allegories*. Elsevier Science Publishers. North Holland.

Goguen, J. (1991). A categorical manifesto. *Mathematical Structures in Computer Science*, 1(1), 9-67.

Jacobs, B. (1999). *Categorical logic and type theory*. Elsevier Science Publishers. North Holland.

Lawvere, F., & Schanuel, S. (1997). *Conceptual mathematics: A first introduction to categories*. Cambridge University Press. UK.

Poya, W. (1992). *An introduction to fibrations, topos theory, the effective topos and modest sets*. (Research Rep. ECS-LFCS-92-208). University of Edinburgh, School of Informatics, Laboratory for Foundations of Computer Science. Edinburgh, UK

Fibration: A basic construct of category theory. Roughly, it is a triple $(\mathbf{C}, \mathbf{B}, \mathbf{f})$ with \mathbf{C} a category (of, say, models), \mathbf{B} another category (of metamodels) called *base*, and $\mathbf{f}: \mathbf{C} \rightarrow \mathbf{B}$ a functor (mapping) between categories (that assigns to each model its metamodel). Given a base object (metamodel) $M \in \mathbf{B}$, those \mathbf{C} -objects (models, schemas) S for which $S \cdot \mathbf{f} = M$ and those \mathbf{C} -arrows between them $s: S \rightarrow S'$ for which $s \cdot \mathbf{f} = \mathbf{id}_M$ (identity mapping of M), form a category (of models in the metamodel M), \mathbf{C}_M . This category is called the *fibre* over M . Fibres are mutually connected in the following way. Let $m: M' \rightarrow M$ be a base arrow (metamodel interpretation) and S be an object over M , $S \cdot \mathbf{f} = M$. Then there is an object S^* over M' , $S^* \cdot \mathbf{f} = M'$, and an arrow $m^*: S^* \rightarrow S$ over m , $m^* \cdot \mathbf{f} = m$, having some remarkable properties (their formulation is too technical to be presented here). In the MMt context, object S^* is the same model S , but its elements are renamed in terms of metamodel M' as governed by the interpretation m , and m^* is the renaming mapping. The technical properties just mentioned abstract this intuition in terms of morphisms in \mathbf{C} and \mathbf{B} . A basic account of fibrations can be found in Poya (1992) or Jacobs (1999); the former is much more lighter technically.⁹

Topos: A category endowed with a certain collection of diagram operations that allow one to perform many manipulations with objects and morphisms. Roughly, in a topos one can perform all constructive operations with objects: intersections, unions, and other analogs of common set theoretical operations, including the powerset (forming the object of all sub-objects of a given object). Typical examples of toposes are the category of sets and mappings between them or the category of graphs and graph mappings.

KEY TERMS

Doctrine: A category endowed with a certain collection of diagram operations that allow one to perform certain logical manipulations with objects and morphisms. For example, in a category called *logos*, one can join and meet objects, take images and co-images of objects under given morphisms, and consider graphs (binary relations) of morphisms. In a *transitive* logos, one can, in addition, consider transitive closures of binary relations. A detailed account of doctrines can be found in Freyd and Scedrov (1990). Like relational algebra is an algebraic counterpart of some version of first-order predicate calculus, a categorical doctrine is a diagram-algebraic counterpart of some sort of logical calculus. Correspondingly, a hierarchy of logical calculi ranging from propositional to first-order predicate to higher-order predicate calculi gives rise to the corresponding hierarchy of categories—doctrines.

ENDNOTES

- 1 Partially supported by Grant 05.1526 from the Latvian Council of Science.
- 2 Details about well-formed keyed sketches and the role of the **KGA** condition can be found in Diskin (1998); see also Diskin and Kadish (2003).
- 3 The choice of names for the internal items of graph $/S^*$ is arbitrary, e.g., add *-superscript to the names used in S .
- 4 One result of this kind is actually well known to the database community: Relational algebra is an algebraic equivalent of a version of first-order predicate calculus.
- 5 For example, the familiar notion of “relational algebra” refers to a certain well-known clone of operations over relations rather than to some specific set of basic operations generating it.

- ⁶ We might call this structure *weak topos*, or *quasi-topos*, or, better, *effective topos*, but unfortunately all these names are already used in CT. To avoid collision, naming this structure *MMt topos* or *MMt doctrine* seems to be reasonable, at least until its place in the hierarchy of doctrines will be precisely defined.
- ⁷ E. W. Dijkstra said these words about elegance in computing on another occasion, but they are just to the point for gMMt.
- ⁸ As a rule, formal mathematical constructs appear within consistent *systems* of concepts rather than discretely. Such systems normally explicate some integral non-trivial intuition, partially embedded in each of the constructs that the system consist of. This way the intuition behind a mathematical construct (like the roots of a tree) can go much wider and deeper than can be seen in the construct as such. The result is that in mathematical modeling, an apt formal counterpart, F , of a real world phenomenon/artifact, W , can offer much more than initially expected from stating the correspondence “ F models W ”. The history of applying mathematics to science and engineering is full of examples when formalisms turned out to be surprisingly clever in their modeling and predictive capabilities. Category Theory, saturated with structures mutually modeling each other so that “everything is interpreted in everything”, is especially rich in such far-reaching integral intuitions. As a result, categorical reformulations of particular applied phenomena/artifacts can be very productive and helpful. A discussion of this phenomenon can be found in Goguen (1991) and Diskin (2001).
- ⁹ **About CT Books:** Category Theory is overly abstract even by mathematical standards, its internal nickname was “the abstract nonsense.” Though CT is extremely rich in ideas on structure engineering, their rigorous presentation inevitably involves a lot of technical details. Therefore, the authors of a CT book for nonmathematicians need to pass between *Scylla* of being comprehensible but limited, and *Charybdis* of being rich in material but overly technical. So far, it seems that no author team sailed safely between the monsters. For example, Lawvere and Schanuel (1997) are quite readable but cover only the basics. On the other hand, Barr and Wells (1995) describe all the constructs we consider in *Math-I* and *Math-II* but the text seems to be too complicated, even though the book is written with a bias towards CT applications in computer science. Hopefully, the MMt interpretation of CT constructs, provided in *Math-I* and *II*, will make the reader’s journey through CT seas more enjoyable.