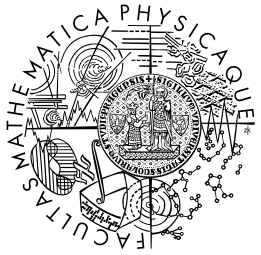


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

DOCTORAL THESIS

Pavel Koupil (Čontoš)

Modelling and Management of Multi-Model Data

Department of Software Engineering

Supervisor of the doctoral thesis: doc. RNDr. Irena Holubová, Ph.D.
Study programme: Computer Science
Study branch: Software Systems

Prague 2022

I declare that I carried out this doctoral thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

First and foremost, I would like to thank my supervisor Irena Holubová for her invaluable advice, constructive feedback and support during my PhD studies. Despite all the complications that the pandemic brought with it, our cooperation was always smooth and friendly.

I also wish to thank all my co-authors. Martin Svoboda for his advice especially during the beginning of my studies. Ivan Veinhardt Latták, Sebastián Hricko, and Jáchym Bártík for their contributions, which took our knowledge a step further. My thanks also belong to all the anonymous reviewers for their constructive comments and suggestions during the publication of the papers.

I would also like to thank to members of L1 Allegra Laser Group from the project ELI Beamlines who allowed me to find the time to pursue my PhD studies.

Last but not least, I am deeply grateful to my wife. Without her tremendous understanding and encouragement over the past few years, it would not be possible for me to complete my studies. We tried our best to support each other in our individual PhD journeys and we shared our joys and sorrows alike.

Dedicated to my father (1958-2016) and ...

Title: Modelling and Management of Multi-Model Data

Author: Pavel Koupil (Čontoš)

Department: Department of Software Engineering

Supervisor: doc. RNDr. Irena Holubová, Ph.D., Department of Software Engineering

Abstract: With the advent of multi-model database management systems, the boundaries of many approaches to data processing were pushed. The aspect of multi-model data introduces a new dimension of complexity and new challenges not seen in single-model systems. We have to address issues arising from the combination of interconnected and often contradictory logical models, such as, e.g., order-preserving/-ignorant, aggregate-oriented/-ignorant, schema-full/-less/-mixed approaches, intra- and inter-model references, intra- and inter-model integrity constraints, or full and partial intra- and inter-model data redundancy. Hence, a number of mature and verified approaches for various data management tasks commonly used for single-model DBMSs cannot be directly applied to multi-model DBMSs.

This thesis aims to propose a new family of unified approaches for both conceptual and logical multi-model modelling and data management. We first analyse the state-of-the-art of related areas. Then we propose abstract data structures to represent multi-model schema and data. These structures are then utilised in the design of approaches for unified schema inference, data migration, schema evolution, and correct backward propagation of changes to the data. All the proposed approaches are implemented and experimentally verified.

Keywords: Multi-Model Data, Conceptual Modelling, Logical Modelling, Schema Inference, Data Migration, Evolution Management, Category Theory

Contents

Preface 3

Commentary 5

0.1 Variety of Data . . . 7
0.1.1 Basic Constructs and Their Unification . . . 8
0.1.2 Multi-Model Data . . . 8
0.2 Category Theory . . . 10
0.2.1 Choice of Category Theory . . . 10
0.2.2 Apparent Similarity to Graph Theory . . . 10
0.2.3 Application of Category Theory in the Proposed Approach 11
0.3 Multi-Model Data Modelling . . . 12
0.3.1 Conceptual Layer . . . 12
0.3.2 Logical Layer . . . 18
0.3.3 Open Questions and Challenges in Data Modelling . . . 33
0.3.4 Contribution: Framework MM-cat . . . 36
0.4 Schema Inference . . . 39
0.4.1 State of the art . . . 39
0.4.2 Closely Related Single-Model Approaches . . . 42
0.4.3 Open Questions and Challenges in Schema Inference . . . 45
0.4.4 Contribution: Framework MM-infer . . . 48
0.5 Evolution Management . . . 50
0.5.1 Closely Related Approaches . . . 50
0.5.2 Open Questions and Challenges in Evolution Management 58
0.5.3 Contribution: Framework MM-evocat . . . 60

1 Categorical Management of Multi-Model Data 63

2 Categorical Modeling of Multi-Model Data: One Model to Rule Them All 65

3 A Unified Representation and Transformation of Multi-Model Data using Category Theory 67

4 A Universal Approach for Multi-Model Schema Inference 69

5 MM-evocat: A Tool for Modelling and Evolution Management of Multi-Model Data 71

Conclusion 73

Bibliography 75

List of Figures 87

List of Tables 89

List of Abbreviations 91

List of Publications **93**

A Category Theory **95**
A.1 Basic Definitions 95
A.2 Functors 98
A.3 Natural Transformations 101
A.4 Universal Constructions 103

Preface

The proposed thesis presents selected results of the author's research in the area of modelling and management of multi-model data. The research has been carried out at the Department of Software Engineering, Faculty of Mathematics and Physics, Charles University in Prague in years 2019-2022. The author is a member of Multi-Model Databases Research Group[1] lead by doc. RNDr. Irena Holubová, Ph.D.

The results are presented as a collection of five selected papers [1, 2, 3, 4, 5] followed by a unifying commentary. Paper I is a vision of a categorical framework, Paper II addresses conceptual modelling of multi-model data, Paper III proposes unifying data structures together with universal schema and data transformation algorithms, Paper IV deals with the inference of a unifying schema from already existing data, and Paper V addresses the problem of schema evolution and its backwards propagation. A complete list of 14 papers – namely, 2 Q1 academic journal articles (2x Journal of Big Data), 3 CORE A conference papers (EDBT 2022, MODELS 2022, MODELS 2021), 5 CORE B conference papers (IDEAS 2022, SAC 2022, ENASE 2022, 2x IDEAS 2021), 1 CORE C conference paper (MEDI 2021), 2 workshop papers (PhD@DASFAA 2021, CoMoNoS@ER 2020), and 1 manuscript under review – is provided at the end of this thesis.

Prior to the summary of the papers, a commentary is provided for each addressed area, namely a motivation, a brief summary of the state-of-the-art, a list of open questions and challenges, and a discussion of our contribution. For convenience, the references to the author's original contribution are marked with a pictogram (see on the right). Finally, we conclude and outline directions of our current and future research. ★

The research included in the selected papers has been supported by several grants, namely the GAČR project no. 20-22276S, and the project GA UK no. 16222 (principal researcher).

Želivec, July 2022

Pavel Koupil (Čontoš)

[1] https://www.ksi.mff.cuni.cz/area.html?id=multi-model-data

Commentary

For decades, relational database management systems (RDBMS) based on the relational model [6] were often the obvious candidate for data management. These robust and time-verified systems were characterised by a schema-first, data-later approach and handled structured data very well. However, nowadays, most of the data consists of very large (***volume***) and varied (***variety***) (un)structured data, which in addition are rapidly generated (***velocity***) and changed (***variability***). Hence, relational databases do not necessarily meet the new data management requirements.

Around the dawn of the second millennium, a new family, the so-called NoSQL database systems [7], has emerged, pushing the boundaries of many approaches to data processing. Compared to the RDBMSs, these systems work much better with complex (un)structured data and respond to changes in data and user requirements, e.g., due to the absence of an explicit schema (data first, schema later/never approach). In addition, the NoSQL systems are often scalable, i.e., they allow us to respond flexibly to the volume of data being processed. Despite all the advantages of NoSQL systems, they are not a replacement for RDBMS, but the two families complement each other appropriately.

However, even the advent of NoSQL systems has not solved all the problems. Currently, one of the most difficult challenges is the ***variety*** of data, i.e., a large number of various data types and formats. For example, based on the structure, data can be classified as structured, semi-structured, and unstructured, and/or based on logical representation, there exists, e.g., *relational*, *array*, *graph*, *key/value*, *document*, and *columnar* data. Besides, in real-world applications the logical models are often combined, overlapped, and linked by references. Hence, the applications deal with so-called *multi-model data*.

In general, approaches that store and process multi-model data can be divided into two groups. The first group consists of (mainly) academia-driven systems, the so-called *polystores* [8], which are based on the idea of *polyglot persistence* [7]. These are a combination of single-model database systems that are managed by a so-called *mediator*, which allows for the use of a single interface. To name just a few representatives, there is, e.g., BigDAWG [9] or Estocada [10]. Alternatively, there exist industry-driven so-called *multi-model database management systems* [11], which support multiple logical models within a single system, where all models are treated as first-class citizens [12]. Obviously, this provides a single interface to work with the data. Currently, there are dozens of representatives of multi-model databases,[2] including originally single-model systems now supporting additional data models [13] or attempts to natively implement multiple data models, such as, e.g., Octopus[3] and ArangoDB.[4]

Although there exists a number of mature approaches for various data management tasks commonly used for single-model DBMSs, most of them cannot be directly applied to multi-model DBMSs. The aspect of multi-model data introduces a new dimension of complexity and new challenges not seen in single-model

[2] https://db-engines.com/en/ranking

[3] https://octopus.com/docs/administration/data/octopus-database

[4] https://www.arangodb.com

systems. We need to address issues arising from the representation of data by a single logical model, as well as from the combination of interconnected and often contradictory models, e.g., (cross-model) references, full and partial (cross-model) data redundancy, and (cross-model) integrity constraints. In general, we lack a family of approaches that focus on:

- *Grasping the contradictory features of different data models.* Ideally, we need a unified (abstract) conceptual representation of the data models that hides the minor differences and puts the corresponding features of the logical models on the same level, allowing us to work with them in a unified way [14].

- *Mutual mapping of conceptual and logical layers.* Having a unified conceptual layer, we need a way to transform this layer into a logical layer. Currently, there exist approaches that transform the conceptual layer into, e.g., a relational model [15]. However, in the case of a combination of multiple logical models, this approach is not straightforward at all, mainly due to the different schema approaches (i.e., schema-full, schema-mixed, and schema-less) and, again, the contradictory features of logical models.

- *Inference of the multi-model schema.* To handle schema-mixed and schema-less data, there exist multiple approaches that infer the implicit schema from already stored data [16]. However, to the best of our knowledge, none of them is generally applicable to multi-model data, i.e., one cannot infer features arising from the combination of multiple models.

- *Unified query language.* Currently, there are many, often non-standardised query languages (not only) for multi-model systems [13], which create a huge burden for the users. The ideal situation is the existence of a single universal and natural language whose expressive power embraces commonly used query constructs and which allows efficient querying over multiple interconnected models.

- *Evolution management and correct propagation of changes.* As user requirements change, the data structures evolve. Hence, we need an approach that is universally applicable to propagate changes to all affected parts of the multi-model system correctly and completely. Moreover, changes in schema and data representation can be exploited to increase the performance of, e.g., querying and other data tasks.

Outline The rest of the thesis is structured as follows: In this Chapter we give an introduction to the selected problems, an overview of open questions and we provide a commentary on our proposed solutions. In Chapter 1 we discuss the vision of a whole multi-model framework. In Chapter 2, we detail our proposed approach to abstract modelling of multi-model data. In Chapter 3 we propose a family of algorithms for data transformation that are independent of the logical representation of the data. In Chapter 4 we describe the proposed algorithm for inference of multi-model schemas. In Chapter 5 we present tools for multi-model schema and data evolution. Finally, we conclude and outline future work.

0.1 Variety of Data

Besides relational databases representing data as relations, NoSQL databases allow us to represent and store data using other models, e.g., as a hierarchical tree data (and structures) or pure graph data. We can classify data models as aggregate-ignorant and aggregate-oriented.

The traditional representative of aggregate-ignorant models is the relational model. This group also includes the array model, which allows to represent spatial data, however with certain limitations (e.g. references between arrays are not supported). An example of a system that implements the array model is, e.g., SciDB.[5] Another aggregate-ignorant representative is the graph model, implemented, e.g., in Neo4j,[6] which allows to represent data in its natural form, i.e., as a system of connected related real-world objects. These connections then allow a completely different querying principle (e.g., graph traversal, neighbourhood search etc.) compared to SQL and its derivatives. Similarly, the RDF model corresponds to a directed graph composed of triple statements.

The simplest representative of aggregate-oriented models is the key/value model implemented in, e.g., Redis[7] and Riak[8] systems. Here, the data is stored as a pair $(key, value)$, with *key* (identifier) referring to *value*, i.e., an object stored in the database as a black box. The document model, implemented in, e.g., MongoDB[9] or MarkLogic,[10] uses a similar principle, i.e., it is also based on pairs $(key, value)$, however, the pairs may form a hierarchical structure (i.e., nesting of pairs is allowed). In particular we refer to the pairs as (unordered) *fields* (JSON) or (ordered) *elements* (XML). Unlike the key/value model, querying and referencing over nested data is allowed. Finally, the column model, implemented, e.g., in Apache Cassandra[11] and Apache HBase,[12] also allows related data to be stored together, but in the form of (optional and possibly structured) pairs $(name, value)$ (i.e., columns) forming rows of column families.

Example 0.1. Figure 0.1 illustrates examples of selected data models. The relational table Customer (purple) represents customers together with their contact details, while the graph model (blue) represents the relationships between customers, i.e., a social network. The key/value pairs (yellow) represent the shopping carts of customers. The document model (green) represents the collection of orders of each customer as a hierarchical document. Finally, the column family Orders (red) represents a list of orders for each customer. Thus, at first sight, we can see that data from a single domain can be suitably represented by different logical models. □

Note that comprehensive descriptions of the data models along with examples are provided in Paper IV. ★

[5] https://www.paradigm4.com
[6] https://neo4j.com
[7] https://redis.io
[8] https://riak.com/products/riak-kv/index.html
[9] https://www.mongodb.com
[10] https://www.marklogic.com/product/marklogic-database-overview/
[11] https://cassandra.apache.org/_/index.html
[12] https://hbase.apache.org

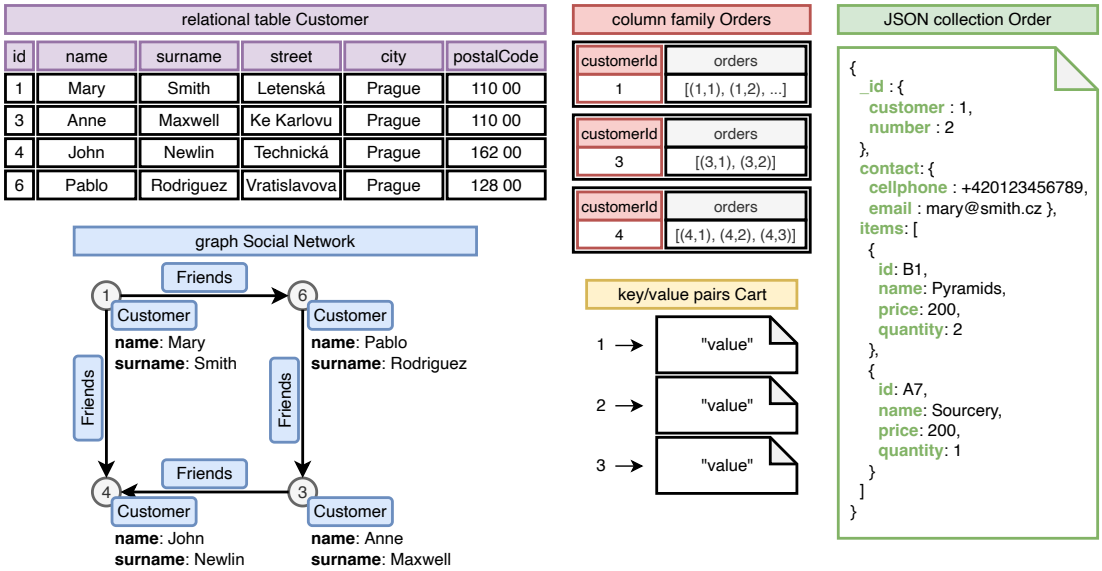


Figure 0.1: An example of variety of data

0.1.1 Basic Constructs and Their Unification

Since the terminology for the constructs of the data models varies greatly, in the following text we will refer to the uniform terminology in Table 0.1. A *kind* refers to a single collection of (possibly similar) instances, corresponding to, e.g., a relational table, a node label, or a collection of documents. A *record* then denotes a single instance of its kind, e.g., a tuple in a relational table, a particular node, or a single document. A record further consists of *properties* such as:

- A *simple property*, e.g., a scalar value.
- A *complex property*, represented, e.g., by a homogeneous[13] or heterogeneous[14] *array*, or a *structure* that contains other properties (possibly both simple and complex).

In addition, complex properties, such as nested documents, form a hierarchy of properties. Hence, a record can be considered as a special kind of a complex root property.

The *domains* correspond to the data types of the *values* of individual properties, while the *active domain* is a set of actively used values. An *identifier* (further distinguished as *simple*, *complex*, and *overlapping*) then uniquely identifies a specific record of a kind. Finally, a *reference* from one kind to an identifier of another kind allows related data to be associated. Note that references are only allowed for certain models.

0.1.2 Multi-Model Data

In general, *multi-model data* is represented by multiple logical models within a single system. Multi-model data not only adopt the properties of single-model data, but in addition:

[13] An array that contains elements of the same type.

[14] An array that contains elements of multiple types. This form of an array is allowed, e.g., in MongoDB document model.

Table 0.1: Unification of terms in popular models

Unifying term	**Relational**	**Array**	**Graph**	**RDF**	**Key/Value**	**Document**	**Column**
Kind	Table	Matrix	Label	Set of triples	Bucket	Collection	Column family
Record	Tuple	Cell	Node / edge	Triple	Pair (key, value)	Document	Row
Property	Attribute	Attribute	Property	Predicate	Value	JSON Field / XML element or attribute	Column
Array	–	–	Array	–	Array	JSON array / repeating XML elements	Array
Structure	–	–	–	–	Set / ZSet / Hash	Nested document	Super column
Domain	Data type	Data type	Data type	IRI / literal / blank node	–	Data type	Data type
Value	Value	Value	Value	Object	Value	Value	Value
Identifier	Key	Coordinates	Identifier	Subject	Key	JSON identifier / XML ID or key	Row key
Reference	Foreign key	–	–	–	–	JSON reference / XML keyref	–

- Analogous to possibly hierarchical models (e.g., the document model), we can connect multi-model data by (1) *inter-model references* or (2) *inter-model embedding* (e.g., a JSONB column in a PostgreSQL table embeds a JSON document into a relational table).
- Similarly to the property labelled graph in Neo4j, we can express *cross-model redundancy*. In this case, we represent the same parts of the data using a combination of data models. We speak about *partial redundancy* if only a subset of the data is represented by multiple models, or a *complete redundancy* if the entire set of data is represented by two or more data models.

The combination of data models within a larger unit (a polystore or a multi-model database) allows us to use the right tool (data model) for the specific tasks. For instance, we represent structured data with small differences in the document model, data containing a large number of relationships between entities with the need for efficient querying over the relationships between entities in the graph model, or fast generated data without the need for complex querying in the key/value model.

Example 0.2. Figure 0.1 also illustrates multi-model data. Compared to Example 0.1, note that there are cross-model references in the data, e.g. from collection *Order* (*customerId*) to table *Customer* (*id*). The data is also redundant, i.e., customer information is stored in both the relational table and the graph. □

0.2 Category Theory

Category theory [17] is a branch of mathematics that provides a way to generalise mathematical structures and the relationships between them. Hence, it is a unifying theory that is useful for finding connections between different areas, not only in mathematics and theoretical computer science.

In this section, we clarify the choice of category theory, indicate the (apparent) similarity between category theory and graph theory, and explain how category theory is applied in the proposed approaches to modelling and managing multi-model data. Note that the basic definitions underlying our proposal, including illustrative examples that are closely related to real-world applications in data modelling approaches, are provided in the Appendix A.

0.2.1 Choice of Category Theory

We have chosen category theory for the unified representation of multi-model data, because it allows for different levels of abstraction and unifies different types of tasks at the abstract level in a natural way. Therefore it has been successfully applied not only for single-model data modelling [18, 19, 20, 21], but also in various (related) areas, such as programming language theory [22, 23], data migration [20], or artificial intelligence (AI) [24, 25] among others. Thus, we do not need to apply a variety of theories and approaches. Instead, combining concepts such as category (see Definition 1), functor (see Definition 6), or natural transformation (see Definition 10) suitably, we are able to represent, e.g., the conceptual and logical schema, the relationship between these schemas, data instances, querying based on pattern matching, data migration, and evolution management.

0.2.2 Apparent Similarity to Graph Theory

At first sight, category theory is similar to graph theory. That is, both categories and (directed) graphs are usually visualised using points and arrows. However, this is where the similarity ends.

A *category* consists of a collection (class) of *objects* and a collection of *morphisms*, where each morphism associates two objects. In addition, (1) morphisms carry a particular meaning, e.g., they define relations or functions between objects, (2) each object is equipped with a so-called *identity* morphism, (3) morphisms are composable using a *composition* operation for which the associative and transitive laws hold, and (4) morphisms and their composition can be compared with each other. Hence, one cannot arbitrarily orient morphisms in a category, but must always respect the composition operation. Finally, categories can, e.g., be mapped, transformed and translated to each other using the notion of *functor*.

In contrast, a *graph* consists of a set of *vertices* (i.e. a special case of a collection) and a set of *edges*, where each edge connects two vertices. Furthermore, edges do not carry any additional meaning[15] and no operations are defined to

[15]However, if edges carry meaning, e.g., a cost, then this is an extended definition of the general graph and its application.

allow edges to be composed or compared. We consider only the notion of a *path* in the graph. In other words, a graph only describes a structure. Finally, graphs can be compared with each other using the notion of *graph homomorphisms*.

However, note that each graph generates a category, referred to as a *free category* (see Definition 5) also known as a *path category*, in which the vertices of the graph form objects and the paths in the graph form morphisms (see Example A.3). In addition, each free category is a *small category* (see Definition 3) and each small category has an underlying graph.

0.2.3 Application of Category Theory in the Proposed Approach

We applied category theory in our approach to multi-model schema and data representation (see Paper II), data migration (see Paper III), and schema and data evolution (see Paper V). ★ ★

The main objective while proposing our approach was to make it user-friendly. Thus, on the one hand, we utilise a complex unifying theory, but on the other hand, we deliberately exploit the apparent similarities with graph theory. For example, we represent the structure of data as a graph (i.e., vertices represent classes of real-world objects, edges represent links between these classes) that freely generates a schema category.

Moreover, to avoid unnecessarily burdening the reader with advanced category theory constructs, e.g., natural transformation (see Definition 10) or universal constructions (see Definitions 13, 14, 16, and 17), we consider these constructs implicitly in our proposal. For example, instead of explicitly using the notions *product* and *coproduct* in the case of the representation of an identifier (i.e., a special case of a *product*) and a set of (overlapping) identifiers (i.e., a coproduct of identifiers), we introduce an internal object (or graph vertex) representation that consists of common notions of a superidentifier and a set of identifiers.

Hence we believe that the model we propose is simple enough that anyone with a basic knowledge of category theory, i.e., the definition of a category (see Definition 1), and functor (see Definition 6),[16] will find our approach easy-to-use.

Although it may seem that we only apply graph theory, we still (implicitly) exploit various levels of abstraction and advanced category theory constructs on which we base the representation of instance data (e.g., functors or natural transformations), data migration (e.g., functors, universal constructions, or natural transformations), schema modification (e.g., functors) and, as future work, querying (e.g., universal constructions or natural transformations).

For the convenience of the reader, all the cases of application of category theory in our approach are summarised in the commentary on data modelling (see Subsection 0.3.4) and the commentary on schema evolution and data migration (see Subsection 0.5.3). Finally, in the Appendix A we also outline in which approaches and for which purpose the above definitions are applied, i.e., we provide additional examples.

[16] Or with a basic knowledge of graph theory, i.e., the definition of a graph and a graph homomorphism.

0.3 Multi-Model Data Modelling

The objective of data modelling is a mapping of real-world objects and their structures to data objects and relationships between them. The conceptual layer captures a generally applicable and platform-independent model of a part of reality. The logical layer is a platform-specific representation of logical data structures in particular (database) systems. Finally, the physical layer organises data into physical units and addresses, e.g., data access.

Currently, there exist a number of approaches for modelling at the conceptual (see Subsection 0.3.1) and logical (see Subsection 0.3.2) layer. However, these are approaches proposed with a relational or graph model in mind as we analyse ★ in [26]. Hence, their general applicability to multi-model data is limited, as the approaches are often unable to capture new structural properties in the data, such as, e.g., relations between properties (note that ER and UML only allow to capture relationships between classes of objects).

In addition, the variety of data models at the logical layer allows to represent data in different ways. However, due to contradictory features of data models, a change in logical representation is not straightforward. Both schema and data loss may occur during this process, e.g. when an order of structural elements carries some information. There have been attempts to unify logical data models [19, 27, 28, 29, 30], but these are often suboptimal solutions that, moreover, cover only a limited subset of existing data models. Although attempts in unification have been made, usually model-specific constructs are inherited [30], thus a broader applicability is still limited.

In the following subsections, we discuss selected existing approaches to modelling at the conceptual and logical level and demonstrate how and if these approaches can be used to represent multi-model data. At the logical level, we mainly focus on approaches that attempt to abstract data models. We compare the selected solutions and based on their analysis we discuss a set of open questions and challenges. Finally, we present a novel multi-model data modelling approach. (Note that in this section we only discuss a static model. Its changes are addressed in a separate section 0.5.)

0.3.1 Conceptual Layer

The objective of conceptual modelling is to represent data without being bound by the features of particular logical models, i.e., we speak about so-called *platform independent modelling* (PIM).

To achieve such a unified abstraction, traditional conceptual modelling languages, e.g., the *Entity-Relationship* model (ER) [31] and the *Unified Modelling Language* (UML) [32] (namely class diagram), suffice with the notions of *entity*, *relation*, *attribute*, *identifier*, and *multiplicity*. Additionally, there also exist approaches [18, 33] based on category theory, or currently less widely used approaches such as the *Natural language Information Analysis Method* (NIAM) [34], representatives of the *Fact-Oriented Modelling* (FORM) [35, 36], or representatives of the *Formal Semantic Database Modelling* [37], e.g., the *Functional Data Model* (FDM) [38, 39], the *Semantic Data Model* (SDM) [40], and the *IFO Model* (IFO) [41]. Last but not least, there also exists an approach [42] allowing to

represent document and graph data at the conceptual layer.

Taking the best of their features, we have proposed a multi-model schema and data abstraction approach [2] inspired by ER and UML and based on category theory. Hence, in this subsection, we mainly discuss the first three mentioned conceptual modelling approaches.

Entity-Relationship Model

The first representative that enables conceptual modelling of complex structures, i.e., real-world entities including their attributes and mutual relationships, is the ER language. The basic constructs are as follows:

- An *entity type* reflects a class of real-world objects. It must contain an identifier and (optionally) includes also additional attributes.
- A *relationship type* represents a binary, n-ary, or reflexive connection between classes of objects. It is implicitly identified by the participants in the relationship, and thus explicit identifier is not allowed. However, similarly to the entity type, the relationship type may contain additional attributes. Moreover, the ER language also allows to name individual roles in a relationship.
- The so-called *weak entity type* is (co-)identified by all other participants of a selected relationship. Note that the ER language lacks the ability to include only selected participants in the weak identifier.
- The *ISA hierarchy* is a possibility to express generalisation (ancestor) or specialisation (descendant) of entity types.
- An *attribute* expresses a characteristic of a class of real-world objects or their relationships in the form of a required, optional, multi-valued, or structured attribute.
- An *identifier* is a special type of an attribute. As such it has identification functionality within the same class of real-world objects. Moreover, identifiers can be categorised based on two criteria:
  - In terms of its structure, we distinguish between a simple identifier (consisting of a single attribute), a composite identifier (consisting of multiple attributes), and an overlapping identifier (i.e., there is at least one attribute that is part of two different identifiers).
  - In terms of entity type membership, a strong identifier (i.e., being directly an attribute of the entity type), an inherited identifier (i.e., a member of the ancestor entity within the ISA hierarchy), a weak mixed identifier (i.e., the entity type is partially identified by an identifier of another entity type with which it enters into a relationship), or a weak external identifier (the entity type is fully identified by an identifier of related entity type) can be distinguished.
- A structured attribute is another special type of a (hierarchical) attribute that can have only trivial depth of 1 (i.e., no further nesting of attributes is allowed).

- A *cardinality*, described as a pair (*min*,*max*), $min \in \{0, 1\}$, $max \in \{1, *\}$, whereas $min \leq max$, expresses multiplicities between an entity type and a relationship type (within a relationship) or between an entity type and its attribute.

The ER language exists in several distinct notations [43], e.g., Chen [31], Reiner et al. [44], Teorey [45], Hoffer et al. [46], and IDEF1X [47], that mutually differ not only visually but also in the constructs used. In other words, there is no standardised format for this language.

Example 0.3. Figure 0.2 illustrates the ER diagram of multi-model data from Figure 0.1. At first glance, the diagram looks complete, i.e., faithfully representing all the features of the data at the conceptual level. For example, there is an entity type *Customer* having two identifiers (*id*) and (*name, surname*), a weak entity type *Order* having a mixed identifier (*id, number*), and the ISA hierarchy between *Product* and its children *Audiobook* and *Book*. Unfortunately, the ER language only allows us to model traditional structured attributes (e.g., *Address*), whereas *Contact* which can be understood as a structured attribute composed of pairs (*name, value*), can only be represented as a binary relationship between *Order* and *Type*. □

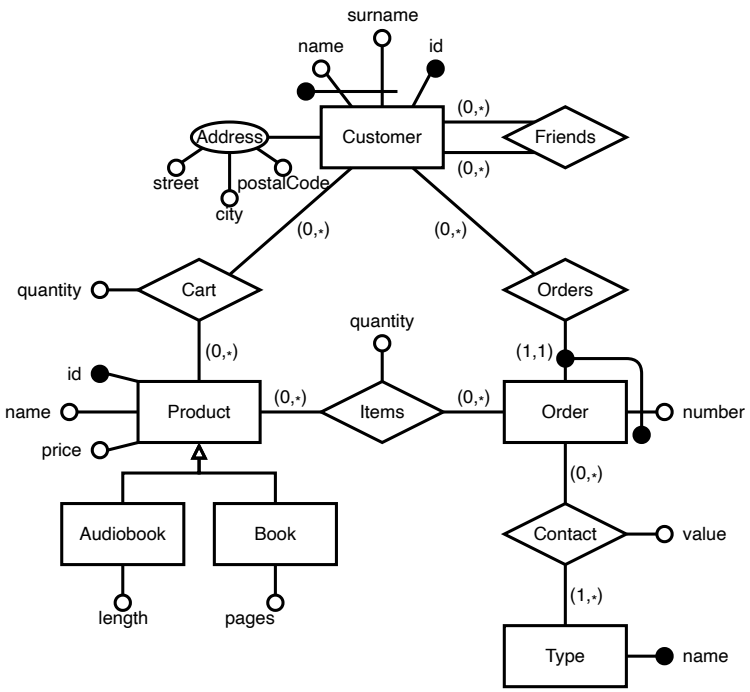


Figure 0.2: An example of ER schema

Unified Modeling Language

Alternatively, the standardised UML allows us to visually represent not only complex structures, but also entire systems. Using the UML, a number of diagrams can be created (i.e., so-called *behavioural* and *structural* diagrams), of which a representative of structural diagrams, so-called *class diagram*, can be used to model the conceptual schema.

The basic constructs of the class diagram are as follows:

- A *concept* is a named set of attributes that represents a class of real-world objects. It corresponds to an entity type in the ER language.
- An *association* is a named relationship (connection) between concepts. It can be defined between two (binary) or more (n-ary) concepts, but also over a single concept (reflexive). In addition, similarly to the ER language, an *association class* can be used to represent the connection between concepts.
- An *attribute* is a logical and untyped data value of a concept or an association. Note that a structured attribute is not an explicit part of the UML, but it can be expressed as an additional concept attached by an association.
- *Multiplicity* expresses the number of instances of concept A associable with an instance of concept B.
- Finally, *inheritance* allows the expression of generalisation (ancestor) or specialisation (descendant) of concepts. Similarly to ER, also *multiple inheritance* is allowed.

The expressive power of the class diagram is limited and does not cover all the important details of the conceptual schema, such as, e.g., identifiers or weak entity types.

Example 0.4. Figure 0.3 illustrates the UML class diagram corresponding to schema of the multi-model data from Figure 0.1. Since UML does not provide a graphical distinction between identifiers and attributes, the identifiers identifying the *Customer* concept cannot be visualised. Moreover, the concept *Order* is not considered as a weak concept. Also note that the structured attribute *Address* is represented as a separate concept that is linked to the parent concept by an association. Furthermore, *Contact* is also represented by an association between the concepts *Order* and *Type*, similarly to the ER model in Figure 0.2. The class diagram also allows us to represent inheritance, specifically the concepts *Audiobook* and *Book* are descendants of the concept *Product*. □

Categorical Conceptual Model (Lippe and Ter Hofstede)

Last but not least, the approach [18] allow to model the conceptual schema and data using an approach based on category theory. The foundation of the approach is a directed multi-graph, so-called *type graph*, that freely generates a category $\mathbf{C}$ (see Definition 5) representing the conceptual schema.

The *type graph* $G = (V, E, L, lbl, pow)$ is a tuple consisting of:

- A set of vertices V, where each vertex $v \in V$ represents a particular type of real-world objects, a relationship type, or an attribute type.
- A set of edges E, where each edge $e \in E$ is an (optionally labelled) directed pair of vertices $e : v_1 \to v_2$, $v_1, v_2 \in V$ determining the way how the vertices participate in various constructions.
- A set of labels $L := \{role, spec, gen, power\_role, elt\_role\}$, where *role* represents a connection between a relationship type and its participant type,

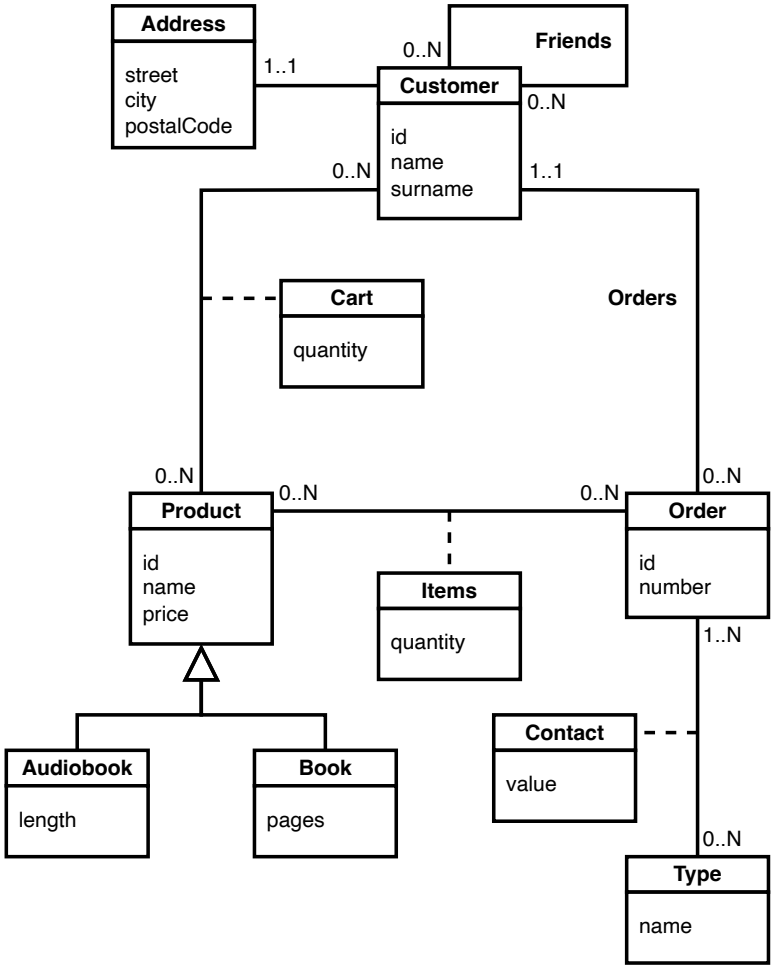


Figure 0.3: An example of UML schema

spec represents a type specialisation, *gen* represents a type generalisation, and *power_role* and *elt_role* representing participants in so-called power type (i.e., a concept of a multi-valued property, e.g., set), the former one representing a connection between a power type and a parent type and the latter one representing the connection between a power type and an element type.

- Function $lbl : E \to L$ associating an edge $e \in E$ with a label $l \in L$.
- Function *pow* is a bijection from edges with label *power_role* to edges with label *elt_role*, which says that an instance of a power type can be identified if and only if its elements are identifiable.
- It must hold that no cycles in the graph are composed of edges with labels *spec* or *gen*.

Moreover, in the conceptual schema $\mathbf{C} = \{\mathcal{O}_\mathbf{C}, \mathcal{M}_\mathbf{C}, \circ, 1\}$ the following holds:

- The uniqueness of an attribute is represented by *monomorphism* $mono \in \mathcal{M}_\mathbf{C}$ (see Definition 4), i.e., each element of $cod(mono)$ determines at most one element in $dom(mono)$.

- A type of a complex identifier is expressed as a product with projections (see Definition 13) to the components of the identifier (see Example 0.5). Moreover, there is a single monomorphism $mono : P \to I$ between an object $P \in \mathcal{O}_{\mathbf{C}}$ corresponding to type of real-world objects $v_p \in V$ and object $I \in \mathcal{O}_{\mathbf{C}}$ corresponds to type of a complex identifier $v_i \in V$.

- A type of a structured attribute is represented similarly to a type of a complex identifier. The only difference is that there is an epimorphism (see Definition 4) $epi : P \to A$ between an object $P \in \mathcal{O}_{\mathbf{C}}$ corresponding to type $v_p \in V$ and an object $A \in \mathcal{O}_{\mathbf{C}}$ corresponds to type of the structured attribute $v_a \in V$. In other words, each value of a structured attribute must be a part of an instance of the parent type.

- The multiplicity of an attribute is expressed as the product $P \times E$ (i.e., a power type), where an object $P \in \mathcal{O}_{\mathbf{C}}$ corresponding to the parent type $v_p \in V$ and an object $E \in \mathcal{O}_{\mathbf{C}}$ corresponds to element type $v_e \in V$. Moreover, it must hold that both projections $\pi_1 : P \times E \to P$, $\pi_2 : P \times E \to E$ are epimorphisms. Note that this approach can be applied to modelling of, e.g., sets, but it is not applicable to represent data collections in general, e.g., arrays (a collection of ordered and possibly duplicate elements), and maps (a sets of pairs (*name, value*)) distinguishable by *name*).

- Inheritance is represented by a complementable monomorphism (see Definition 15) corresponding to an edge with a *spec* label. In other words, each instance of a child must correspond to a unique instance in each of its ancestors. Also note that multiple inheritance is allowed. Moreover, the subtype diagram commutes, i.e., the children have an access to attributes of theirs ancestors.

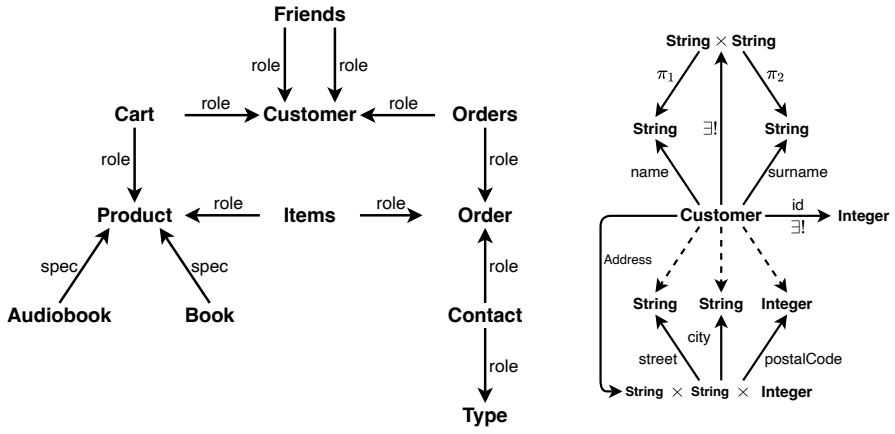
- Generalisation of objects $A, B \in \mathcal{O}_{\mathbf{C}}$ is represented as pushout $A + B$ (see Definition 17).

Finally, the approach also allows to represent data instances conceptually. The foundation for data representation is the so-called *instance category*, which can be based on the categories of sets **Set**, finite sets **FinSet**, partial sets **PartSet** (allowing to represent missing values, i.e., null), relations **Rel**, etc. [18].

Example 0.5. Figure 0.4 illustrates a conceptual categorical schema of the multi-model data from Figure 0.1 represented using approach [18]. For the sake of clarity, we do not show the identity morphisms and we divided the schema into two parts: Figure 0.4 (a) depicts only types of real-world objects and their relationships. Figure 0.4 (b) depicts the attributes of type *Customer*.

Note that relationships (i.e., *Friends*, *Items*, *Cart*, *Orders*, and *Contact*) are modelled as a product with *role*-labelled projections to the participants of the relationship. On the other hand, inheritance is modelled as a co-product with *spec*-labelled inclusions (i.e., the child is a specialisation of the parent). Figure 0.4 (b) illustrates the representation of identifiers using monomorphisms (explicitly labelled with an existence quantifier). In other words, there is only a single mapping between type *Customer* and its *id*. The complex identifier (*name, surname*) is represented as a product with projections to attributes *name* and *surname*. There

is once again a single mapping between *Customer* and the complex identifier. Finally, a structured attribute is represented as a product with projections, but in this case $Address : Customer \to String \times String \times String$ is an epimorphism. Also note that composition $street \circ Address$ allows direct access from *Customer* to *String*. □



(a) (b)

Figure 0.4: An example of a conceptual schema (Lippe and Ter Hofstede)

Comparative Summary

Table 0.2 summarises the expressive power of the three described approaches. In addition, it involves also a comparison of our approach inspired by them and described in detail in Paper II. ★

In principle, all the presented approaches allow for modelling of the traditional concepts of real-world objects, relationships, and their attributes. They differ mainly in their ability to model identifiers, where the UML class diagram does not allow to specify an identifier[17] and, therefore, it does not even work with the principle of a weak entity type partially or completely identified by another entity type. Note that in ER we can model an identifier as a special type of an attribute and that in the categorical approach we use monomorphisms. Moreover, the categorical approach considers an explicit identifier of a relationship type, whereas in the ER language the relationship type is identified implicitly by the participants of the relationship. Finally, the selected approaches differ in the possibility of expressing a structured attribute. The categorical approach allows expressing a structured attribute with unlimited depth and unrestricted structure, while the ER language allows only structured attributes of trivial depth and the UML class diagram expresses structured attributes using a concept and an association.

0.3.2 Logical Layer

While at the conceptual level we view the data in a platform-independent way, the objective of so-called *platform-specific modelling* (PSM) is to capture often

[17] UML allows us to textually express integrity constraints, including the identifier, using the Object Constraint Language (OCL) [48].

Table 0.2: Expressive power of approaches modelling conceptual layer

	ER [31]	**Class diagram (UML)** [32]	**Lippe and Ter Hofstede** [18]	**MM-cat** [3]
Object class	Entity type	Concept	Object	Object
Relationship class	Relationship type	Association (class)	Product + projections (role)	Product + projections
Simple attribute (property)	Attribute	Attribute	Epimorphism to object	Epimorphism to object
Map-like property	No	No	No	Product + projections
Multiplicity	Cardinality (Chen)	Multiplicity	Product + projections (power_role, elt_role)	Product + projections
Role	Role	Named association	No	No
Inheritance	ISA hierarchy	Inheritance	Morphism (*spec*)	Object + monomorphism
Generalisation	Multiple ISA hierarchy	Multiple inheritance	Injection (*gen*)	Injection
Structured attribute (property)	Structured attribute	Concept + association	Product + projections	Product + projections
Reflexive relationship class	Reflexive relationship type	Reflexive association	Object + morphisms (*role*)	Object + morphisms
N-ary relationship class	N-ary relationship type	N-ary association (class)	Product + morphisms (role)	Product + projections
Weak object class	Weak entity type	No	Object + monomorphism	Object + monomorphism
Identifier	Identifier	No	Monomorphism	Monomorphism
Complex identifier	Complex identifier	No	Monomorphism to product	Monomorphism to product
Multiple identifiers	Yes	No	Yes	Coproduct of products
Overlapping identifier	Overlapping identifier	No	No	Overlapping products
Relationship identifier	Implicit	No	Implicit	Implicit / Explicit
Integrity constraints	Identifier	OCL	Identifier	Identifier, Reference

non-transferable characteristics of particular database systems. For example, we consider particular data structures, i.e. graphs, trees, and matrices implemented by the actual DBMS representatives,[18] as well as academic proposals (e.g., the X-SEM model [49] for XML data).

In this subsection, we discuss selected existing approaches that address the unification of different logical models, in particular category theory based approaches [19, 20, 21], the NoSQL abstract model (NoAM) [27], associative arrays (AA) [28], the tensor data model (TDM) [29], and the U-Schema [30]. We also verify if the approaches are generic enough to cover various characteristics of popular data models and multi-model data in general.

[18] Various popular models based on data structures are discussed in detail in Paper IV. ★

Categorical Graph-Oriented Object Data Model (CGOOD)

The approach [19], which is based on the *Graph-Oriented Object Database Model* (GOOD) [50], enables us to work with object and relational data in a unified way (i.e., it aims to abstract these two approaches at the logical level). Abstract data instances are represented as typed graphs, with schema and data defined solely in terms of categorical constructs. In addition, this approach allows graph pattern matching, which further provides a basis for evolution management and querying.

The core structure of the CGOOD model is the directed graph $G = (V, E, srt, tgt)$, which represents both the schema (i.e., so-called *typegraph*) and the data. An object (from the object model), a corresponding tuple (from the relational model), or an active domain of a property is represented by a vertex $v \in V$, whereas the property (i.e., the fact that a certain value is a property of a complex object) is represented by an edge $e \in E$. Note that an edge also represents a function between two types of objects, e.g., *getter*, *isAncestor* (isa), etc. Finally, functions $src, tgt : E \to V$ assign the source and target vertices to the edge accordingly.

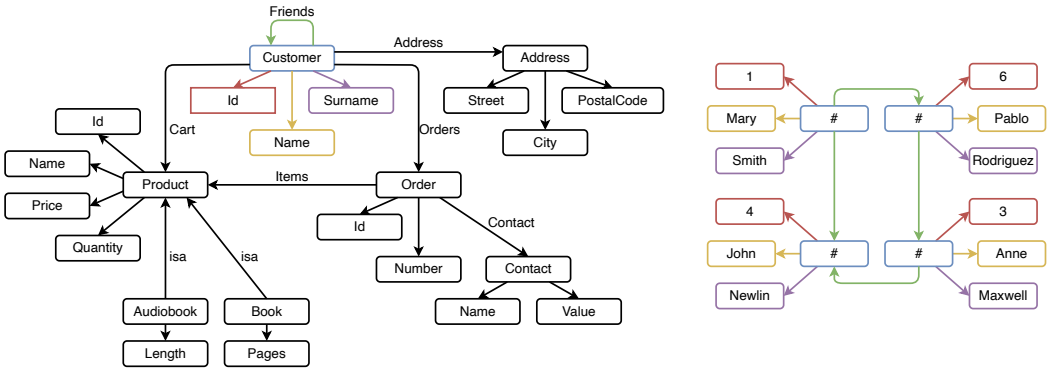
Categorically speaking, a particular graph G is represented as a set-valued functor from Example A.5. Moreover, a collection of such graphs forms a category of all graphs $\mathbf{G} = (\mathcal{O}_\mathbf{G}, \mathcal{M}_\mathbf{G}, \circ, 1)$, which corresponds to the functor category from Example A.8. The category $\mathbf{G}$ also provides a fundamental framework for the definition of a data instance. A so-called *typed instance* is a morphism $Inst : G \to T$, where $Inst \in \mathcal{M}_\mathbf{G}$, and $G, T \in \mathcal{O}_\mathbf{G}$. Note that $Inst$ corresponds to a graph homomorphism between G and T, i.e., there exists a mapping $Inst_V : V_G \to V_T$ specifying the type of the value, and a mapping $Inst_E : E_G \to E_T$ preserving the structure of G.

Example 0.6. Figure 0.5 (a) illustrates a typegraph representing the logical schema of the multi-model data from Figure 0.1. Figure 0.5 (b) depicts a part of the data. For easier understanding, the vertices and edges of the typegraph are labelled by strings, though this is not necessary from a categorical perspective (CGOOD works with unlabelled graphs). In addition, we use colours to represent the mapping of the data to the corresponding types, i.e., values 1, 3, 4, 6 are mapped to *Id*, values Mary, Anne, John, Pablo are mapped to Name, etc. Note that the mapping preserves the structure, i.e., if there exists an edge between two vertices in the data, then there will be an edge between the corresponding vertices in the typegraph. Finally, note that the data does not contain explicit identifiers. □

To conclude, note that category theory in the context of relational and object-relational models has also been addressed in the works [18, 51, 52, 53].

Categorical Logical Model (Spivak et al.)

The approach [20] represents the relational database schema as a small category (see Definition 3) and corresponding data instance as a functor (see Definition 6). The authors also propose a category of all schemas and allowed operations between the schemas. Hence, in combination with so-called *data migration functors* built on top of the schema operations, the migration of data instances is allowed. Moreover, these data migration functors form the basis of a categorical query lan-



(a) (b)

Figure 0.5: An example of CGOOD application to multi-model data

guage CQL [54][19] over relational data. Finally, this approach allows to convert relational data to RDF triples [55] and vice versa.

However, in order to apply this approach, the authors presume that the database (i.e., a set of named kinds) is in the so-called *categorical normal form*, defined as follows:

- Each kind t contains a simple identifier id_t.
- For each additional property c of kind t, $c \neq id_t$, there is a target kind t' such that each record of kind t is mapped to exactly one record of type t', i.e., there is a mapping $c : t \to t'$.
- If two paths $p : t \to t'$, $q : t \to t'$, $p \neq q$ represent the same mapping of records between two kinds, the equivalence $p \simeq q$ must be included in the schema.

In addition, the authors define the notion of a categorical path equivalence relation (CPER) on the graph $G = (V, E, src, tgt)$ denoted by $\simeq$, which has the following properties:

- If p,q are paths of the graph G and $p \simeq q$, then $src(p) = src(q)$.
- If p,q are paths of the graph G and $p \simeq q$, then $tgt(p) = tgt(q)$.
- Let $p, q : b \to c$ be paths and $m : a \to b$ be an edge in the graph G. If $p \simeq q$ holds, so does $p \circ m \simeq q \circ m$.
- Let $p, q : a \to b$ be paths and $n : b \to c$ be an edge in the graph G. If $p \simeq q$ holds, so does $n \circ p \simeq n \circ q$.

Finally, the schema of a database in categorical normal form is a pair $(G, \simeq)$, where $G = (V, E, src, tgt)$ freely generates the category $\mathbf{G} = (\mathcal{O}_\mathbf{G}, \mathcal{M}_\mathbf{G}, \circ, 1)$ as follows:

- $\mathcal{O}_\mathbf{G}$ correspond to the vertices in V.

[19] https://www.categoricaldata.net

- $\mathcal{M}_{\mathbf{G}}$ correspond to the paths in the graph G composed of edges E.
- $\circ$ corresponds to the path concatenation operation.
- $\simeq$ is a categorical path equivalence relation on G.

In other words, the kinds (i.e., tables) in the schema are determined by vertices $v \in V$ mapped to objects $O \in \mathcal{O}_{\mathbf{G}}$, the properties (i.e., columns) are determined by edges $e \in E$ mapped to morphisms $m \in \mathcal{M}_{\mathbf{G}}$, and the integrity constraints are represented by a categorical path equivalence relation. A closer look allows us to further divide the objects into (1) objects $T \in \mathcal{O}_{\mathbf{G}}$, each corresponding to a kind of a single property (i.e., a single one-column relational table) and (2) objects $D \in \mathcal{O}_{\mathbf{G}}$, each corresponding to a generic data type, e.g. String, Number, Boolean, etc. As for morphisms, we can distinguish: (1) identity morphisms $id_t : T \to T$, each modelling an identifier of a specific kind (i.e., relational table); in the case where the kind has no identifier, id_t is considered implicitly, (2) identity morphisms $id_d : D \to D$, each modelling an identifier for a domain of a particular data type, (3) morphisms $r : T \to T'$, each modelling a reference from T to T', and (4) morphisms $c : T \to D$, for each kind T and its property (distinct from the identifier and reference) of data type D. Also note that the categorical normal form does not support complex and overlapping identifiers.

The authors also introduce a *set-valued functor* $Inst_{\mathbf{G}} : \mathbf{G} \to \mathbf{Set}$ representing a particular instance conforming to the schema $\mathbf{G}$ (see Example A.5).

Example 0.7. Figure 0.6 (a) illustrates the multi-model schema from Figure 0.1 represented by the approach [20]. Objects depicted using a solid line represent particular tables (e.g., Customer), while objects depicted with a dashed line represent data types (e.g., String). Although the schema appears to faithfully capture the logical schema of multi-model data, it is in fact expressed in terms of interconnected single-column (non-aggregated) relational tables. Moreover, this approach does not consider complex or overlapping identifiers, but only trivial one-column identifiers. Also note that instead of an ISA hierarchy, the specialisation of a child is represented by addition of the property *Type* to the kind *Product*, as the approach [20] does not allow explicit modelling of the ISA hierarchy.

Finally, the part of the instance is illustrated in Figure 0.6 (b). □

Papers [56, 57] extend this approach to support multiple logical models, i.e., not just the relational model. However, the proposal only considers separate models (namely relational, graph, and document) over which the data can be queried and migrated between using functors and natural transformations.

Algebraic Property Graph (APG)

Yet another approach [21] is based on type theory, algebra, and category theory, i.e., it is closely related to algebraic databases [58], and aims to cover the area of property graphs. The authors propose the so-called *algebraic property graph* (APG) to represent a general property graph. In addition, a set of rules for transforming selected data models (i.e., relational, RDF, key/value, XML and, JSON document) into the APG representation is proposed. Also, the basic operations of

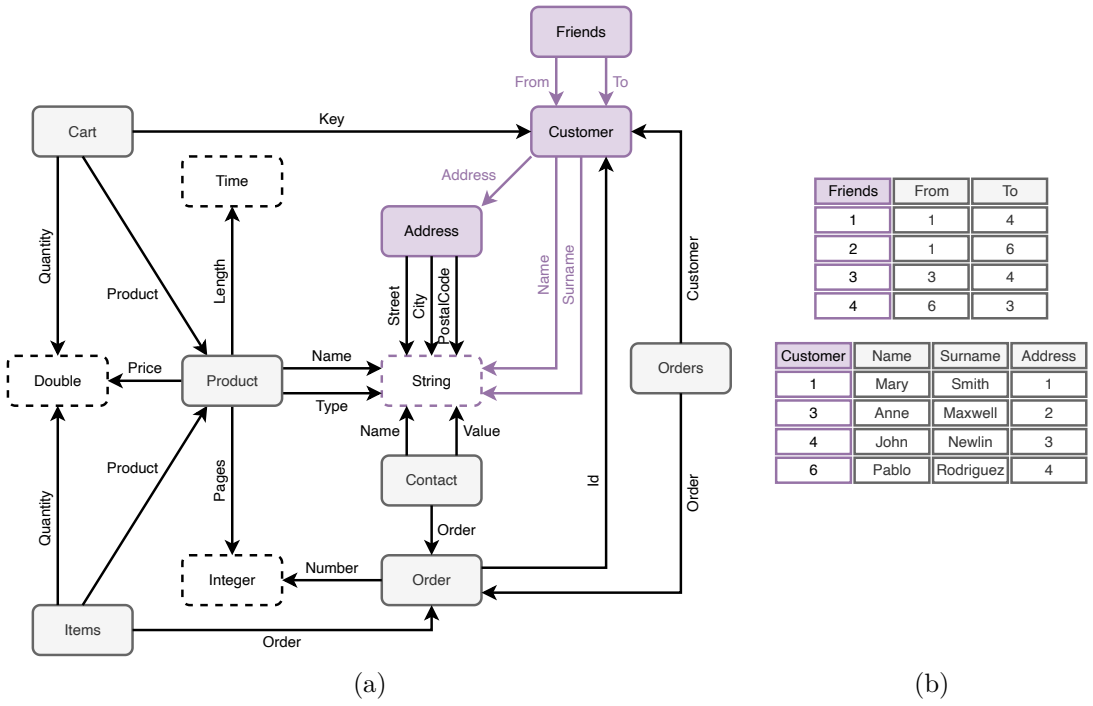


Figure 0.6: An example of a logical categorical schema and data (Spivak et al.)

querying and migration between two APGs are proposed. The whole framework is implemented as a part of an open-source tool CQL.[20]

Property graph $G = (V, E, L, src, tgt, lbl_v, lbl_e)$ is defined as follows:

- V is the set of vertices of the graph G.
- E is the set of edges of the graph G.
- Functions src, tgt realize the edge orientation.
- Unlike the ordinary graph, G also contains a set of labels L and functions $lbl_v : V \to L$, $lbl_e : E \to L$ that assign labels $l \in L$ to vertices $v \in V$ and edges $e \in E$ accordingly.

Note that internally each vertex and edge contains a unique identifier and a set of key/value pairs called properties.

The authors represent the property graph as a category $\mathbf{A} = (\mathcal{O}_\mathbf{A}, \mathcal{M}_\mathbf{A}, \circ, 1)$, where $\mathcal{O}_\mathbf{A} = \{L_\mathbf{A}, T_\mathbf{A}, E_\mathbf{A}, V_\mathbf{A}\}$ and $\mathcal{M}_\mathbf{A} = \{v_\mathbf{A}, \tau_\mathbf{A}, \lambda_\mathbf{A}, \omega_\mathbf{A}\}$ such that:

- $L_\mathbf{A}$ represents the set of labels to be assigned to each vertex of the graph.
- $T_\mathbf{A}$ denotes the set of types that can be assigned to the vertices of the graph. Each type $t \in T_\mathbf{A}$ is a term of the grammar $t := 1 \mid p \mid l \mid t_1 + t_2 \mid t_1 \times t_2$, where 1 is a type of a trivial value, $p \in P$ is a primitive type, $l \in L_\mathbf{A}$, $t_1 + t_2$ is a complex type (e.g., union type), and $t_1 \times t_2$ is a type of an edge.
- The set of elements $E_\mathbf{A} = V \cup E$ represents the elements of the graph G, i.e., the vertices and edges.

[20] https://www.categoricaldata.net

- The set $V_{\mathbf{A}}$, where the elements $v_i : t_i$ are terms of the typed grammar $v : t := () : 1 \mid inl_{t_2}(v : t_1) : t_1 + t_2 \mid inr_{t_1}(v : t_2) : t_1 + t_2 \mid (v_1, v_2) : t_1 \times t_2 \mid v_p : p \mid e : \lambda_{\mathbf{A}}(e)$, where $t, t_1, t_2 \in T_{\mathbf{A}}$, $p \in T_{\mathbf{A}}$ is a primitive type, and $e \in E_{\mathbf{A}}$.
- The function $v_{\mathbf{A}} : E_{\mathbf{A}} \to V_{\mathbf{A}}$ assigns a value to each element.
- The function $\tau_{\mathbf{A}} : V_{\mathbf{A}} \to T_{\mathbf{A}}$ associates a data type with each value.
- The function $\lambda_{\mathbf{A}} : E_{\mathbf{A}} \to L_{\mathbf{A}}$ attaches a label to each element.
- The function $\omega_{\mathbf{A}} : L_{\mathbf{A}} \to T_{\mathbf{A}}$ determines the (data) type of each label.
- The structure of the graph must correspond to its schema, i.e., $\tau_{\mathbf{A}} \circ v_{\mathbf{A}} = \omega_{\mathbf{A}} \circ \lambda_{\mathbf{A}}$.

A particular algebraic property graph (i.e., an instance) is then a functor $F : \mathbf{A} \to \mathbf{Set}$ (see Example A.5).

Example 0.8. An example of $l \in L_{\mathbf{A}}$ is, e.g., `Customer` attached to a vertex representing a particular customer, and `name` attached to an edge that assigns a name to a customer.

Examples of $t \in T_{\mathbf{A}}$ include `Customer` (i.e., the type of vertex attached by label `Customer`), `String` (i.e., the type of a vertex that represents an attribute), `Customer` $\times$ `String` (i.e., the type of an edge that assigns a name of type String to the Customer), and `Product + Audiobook` (i.e., the type of vertex that has multiple labels attached).

An example of a typed value is, e.g., `"Mary":String`, `():1` (i.e., a trivial value of an arbitrary vertex), `(`c_1`, "Mary"):Customer` $\times$ `String` (i.e., an edge value, where c_1 is a reference to the vertex corresponding to the customer with $id = 1$).

Then the function $\lambda_{\mathbf{A}}(c_1) = Customer$ assigns the label Customer to the vertex c_1. The function $v_{\mathbf{A}}(n_1) = (c_1, "Mary")$ determines the value of the edge referenced by the reference n_1, or $v_{\mathbf{A}}(c_1) = ()$. Examples of exploitation of the function to determine the label of a graph element are $\omega_{\mathbf{A}}(Customer) = Customer$, and $\omega_{\mathbf{A}}(name) = Customer \times String$. Finally, an example of determining the type of a typed value is $\tau_{\mathbf{A}}("Mary", String) = String$. □

Example 0.9. Figure 0.7 (a) illustrates the schema of the data from Figure 0.1 represented as APG. The green objects correspond to real-world objects, the black and white objects represent property values, and the blue arrows, each crossing a blue object, represent directed relationships between the objects. Note that properties of objects or relationships are represented by the edge leading to the black and white objects.

Figure 0.7 (b) illustrates a part of the multi-model data from Figure 0.1 that corresponds to the schema in Figure 0.7 (a). Note that the approach allows us to represent the ISA hierarchy by attaching multiple labels to a single vertex. Moreover, vertices are identified only by a simple implicit identifier, i.e., complex identifiers are not allowed. Finally, the structured attribute Address is inlined to the Customer. □

APG is also suitable for data and schema migration. As every instance of APG corresponds to a functor $F_{1,2} : \mathbf{A} \to \mathbf{Set}$, the migration between two instances of

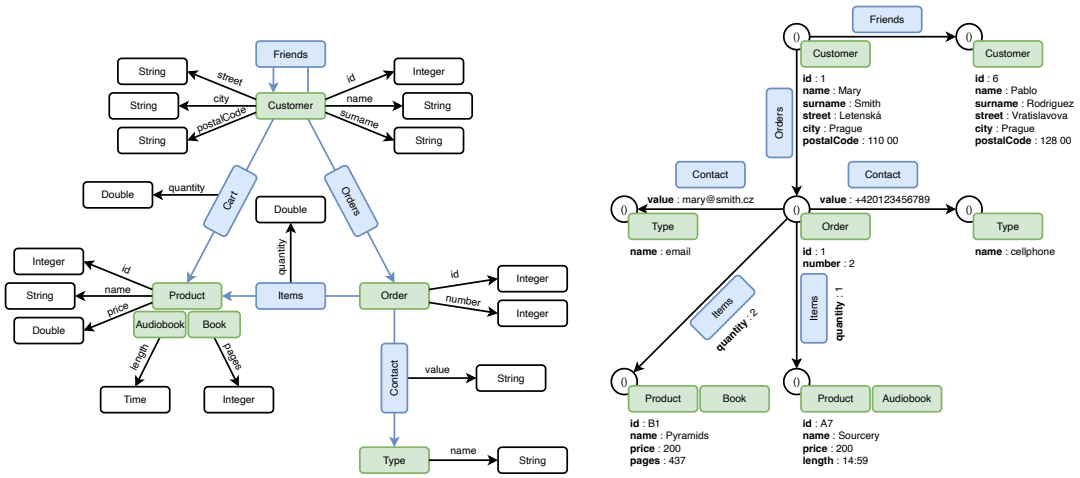


Figure 0.7: An example of APG (a) corresponding to data (b)

APGs can be formally described as a natural transformation $\alpha : F_1 \rightarrow F_2$ (see Definition 10).

Finally, the authors also propose a method for APG to represent the schema of other data models (i.e., relational, RDF, key/value, XML and JSON document), but do not cover all the properties of the models, e.g., uniqueness of values, order of properties, structured and overlapping identifiers.

NoSQL Abstract Model

The approach [27] is a system-independent model of aggregate-oriented NoSQL database systems, i.e., it covers key/value, document, and columnar models. In addition, it is designed for performance, scalability and data consistency.

The basic constructs of the proposed abstract model are as follows:

- The NoAM database $D = \{C_1, \ldots, C_n\}$ is a set of collections.
- Each collection $C = (id_C, B)$ is uniquely identified by the collection key id_C and contains the set of blocks $B = \{b_1, \ldots, b_m\}$. Examples of collections include bucket, document collection, and column family.
- Each block $b = (id_b, E)$ is uniquely identified by the block key id_b and contains a non empty set of entries $E = \{e_1, \ldots, e_p\}$. Each block corresponds to an aggregate, e.g., a single key/value pair, a document, or a row of a column family. A block is also the largest data unit for which atomic operations are considered.
- Entry $e = (e_k, e_v)$, where e_k uniquely identifies an entry within a block and e_v represents a primitive or complex value. An entry corresponds to a document field and a table column.

NoAM employs two data representation strategies, i.e., *input per each top-level property* (ETF) and *input per each aggregated object* (EAO).

When applying the ETF strategy, each block b represents a single record, wherein the block key id_b corresponds to the identifier of the record, and the set

of entries E contains one pair (e_k, e_v) for each property except for the property having the identification feature. If the top-level property represents a complex property, i.e., it is not an atomic property, this property is also represented as an entry and not as a set of nested entries (e.g., a nested block).

Alternatively, when applying the EAO strategy, each record is represented by its own block, where the block key id_b corresponds to an identifier and the set of entries is constituted by a single entry (e_k, e_v). Note that $e_k = \epsilon$ and the value is the aggregate corresponding to the record without the identifier (as it is already used as id_b).

Example 0.10. Figure 0.8 (a) illustrates the application of the ETF strategy to the kinds *Order* and *Customer* from Figure 0.1, while Figure 0.8 (b) illustrates the application of the EAO strategy to the same data. A comparison of the two strategies demonstrates that the block identifiers are the same and the blocks differ only internally. While the block contains only a single complex entry identified by the (meta)value ϵ when using the ETF strategy, the block consists of a set of uniquely recognisable entries when using the EAO strategy. However, even if using the EAO strategy, the entries corresponding to a complex structure are not split further into smaller units (e.g., see properties contact and items). □

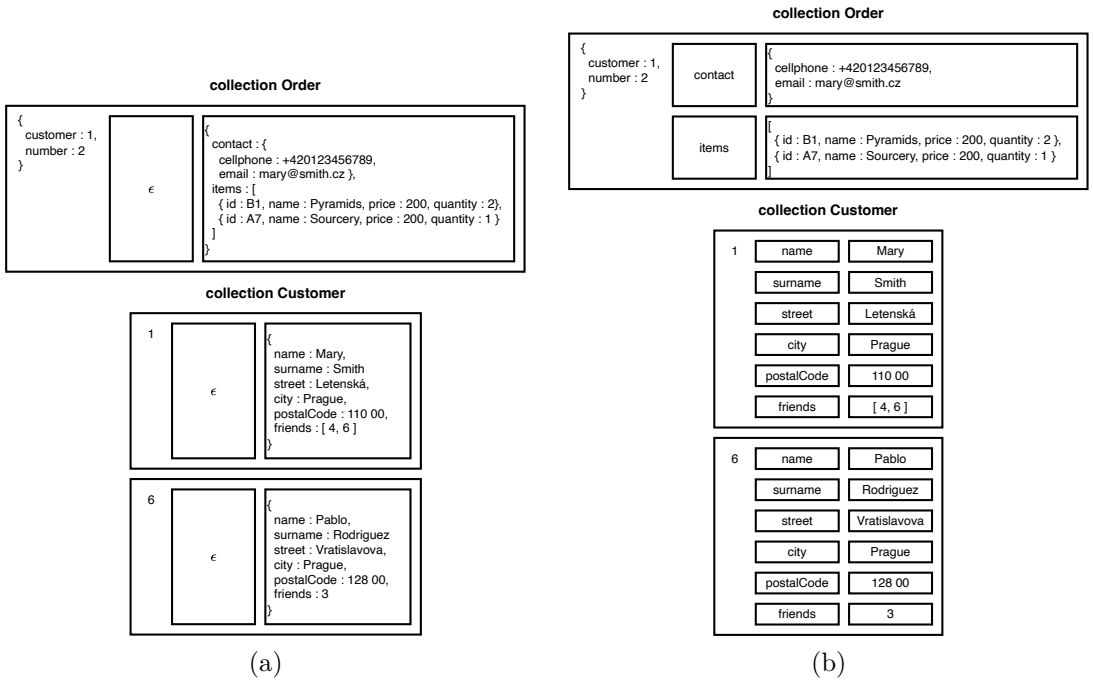


Figure 0.8: An example of (a) ETF and (b) EAO strategies

In addition, the so-called access paths ap are considered, allowing to represent the complex property p as a set of entries $\{(ap_1, v_1), \ldots, (ap_n, v_n)\}$. As such, each nested property c is represented as an entry (ap_c, c), where the entry key ap_c is the sequence of steps required to access property c from parent p, and the value c is the accessed nested property.

In order to select an appropriate aggregate representation strategy, the authors propose a set of rules for partitioning the data model. As the authors suggest, the chosen strategy should reflect, e.g., data access patterns and support

strong consistency and efficient execution of update operations. Unfortunately, the approach does not consider general multi-model data, i.e., a combination of possibly overlapping data models, but only aggregate-oriented data. Note also that the approach only allows for so-called embedding of kinds (i.e., nested complex properties representing an additional structure), but it does not consider references between different kinds (i.e., across different data collections) at all.

Associative Arrays

The next approach [28] is a system-independent model that allows selected data models to be represented uniformly at the logical and physical layers. As such, it is based on a generic data structure, so-called *associative arrays*.

The core of the approach is the associative array A, i.e., a mapping of a two-dimensional key to the value $A : K_1 \times K_2 \to \mathbf{V}$, where:

- Key K_1 corresponds to an array row.
- Key K_2 corresponds to an array column.
- A domain of a key can be an arbitrary set of ordered values, e.g., integers, strings, etc.
- The values $k \in K_i$ must be unique within the key K_i, i.e. there does not exist $k_i, k_j \in K : k_i = k_j$, but there may exist two keys $k_1 \in K_1$, $k_2 \in K_2$ such that $k_1 = k_2$.
- There is no row or column that is completely empty, i.e., an associative array does not allow us to represent an empty record or a property having an empty active domain.

Associative arrays allow to represent both aggregate-ignorant and aggregate-oriented models. In the former case, the models are represented as a matrix where the row key corresponds to the recorder identifier and the column key corresponds to the property name, while in the latter case it is a sparse matrix that additionally represents a hierarchical arrangement of data in the keys. For example, the row key contains not only the document identifier, but additionally a unique index for nested array elements to determine their order, and the column key reflects the hierarchy between properties, i.e., top-level properties are represented by the property name, while further nested properties are represented by the minimum sequence of steps required to access the property (similarly to the access paths in NoAM). However, the approach does not model, e.g., references, multiple (possible overlapping) identifiers, and the ISA hierarchy at all.

Example 0.11. Figure 0.9 illustrates the application of the approach [28] to the multi-model data from Figure 0.1. As illustrated, the approach allows to represent heterogeneous models as a disjunctive set of associative fields. Note that the approach allows to represent a complex identifier or complex and nested properties, including capturing the order of elements in the array, as illustrated by associative array representing kind *Order*. □

The proposed approach also provides querying and transformations. It uses basic operations, e.g., element-wise addition, element-wise multiplication, and

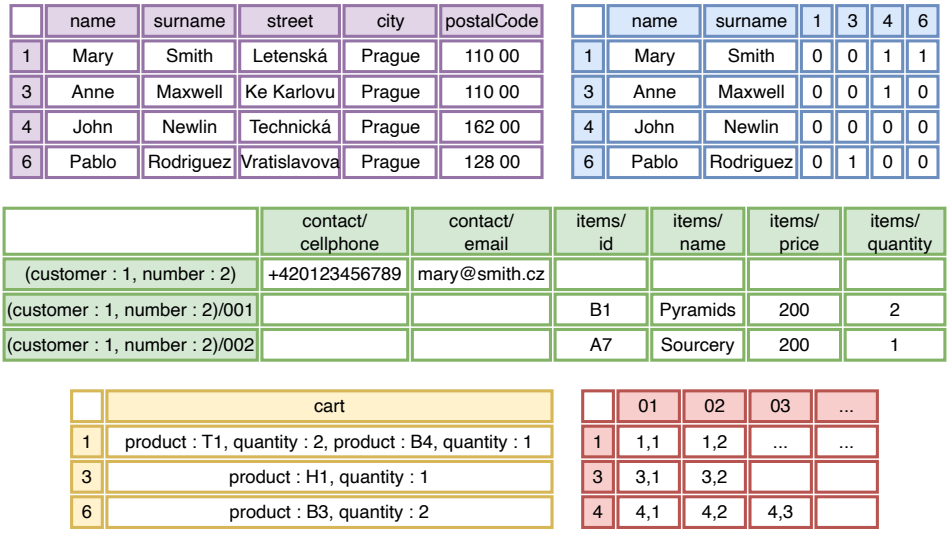


Figure 0.9: An example of Associative Arrays

array multiplication, which correspond, e.g., to the database operations of table union, intersection, and transformation.

Tensor Data Model (TDM)

Last but not least, the approach [29] allows us to represent multi-model data in terms of tensors [59], i.e., the following matrix generalisation: 0th order tensor is a scalar, 1st order tensor is a vector, 2nd order tensor is a matrix, and n-th order tensor is so-called *higher-order tensor*.

In TDM, tensors are defined as the mapping $T : K_1 \times \cdots \times K_n \to \mathbb{V}$, where: $n \in \mathbb{N}$ is the order of the tensor, K_i is the dimension of the identifier $K_1 \times \cdots \times K_n$, and $\mathbb{V}$ is a set of values. In addition, tensors are unambiguously named and typed.

Tensors are used in three possible ways in TDM:

1. Associative arrays, denoted by A_i for $i = 1, \ldots, n$, model dimensions of a tensor X. Such arrays have only one set of keys associated with integers using bijective function $A_i : K_i \to \mathbb{N}$.

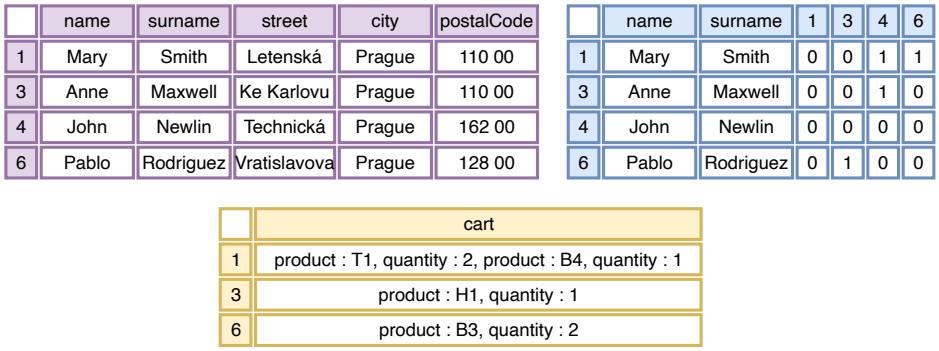
2. At a lower level, an associative array is used to represent the values of a sparse n-order tensor by associating compound keys from dimensions to values $A_{vst} : K_1 \times ... \times K_n \to \mathbb{V}$.

3. For tensors with non-numerical values, two associative arrays are used: (i) to map keys dimensions to a set of integer keys (A_{vst}) and (ii) to map the integer keys to non-numeric domains values (one integer is associated with each different value).

Operations with tensors are analogous to operations with matrices and vectors, e.g. multiplication, transpose, unfolding (transformation of a tensor into a matrix), factorisation (decomposition), etc.

Example 0.12. Figure 0.10 illustrates the application of the approach [29] to the multi-model data from Figure 0.1. At first sight, the approach is identical to associative arrays proposed in [28]; however, the document and column models are

not considered. That is, a hierarchy between individual properties and complex properties such as nested documents, arrays, etc. cannot be represented. On the other hand, this approach allows us to apply higher-order tensors, e.g., to represent n-ary relations. In this case, we could use a third-order tensor to represent, e.g., graph data illustrating relationships between two customers. □



	name	surname	street	city	postalCode
1	Mary	Smith	Letenská	Prague	110 00
3	Anne	Maxwell	Ke Karlovu	Prague	110 00
4	John	Newlin	Technická	Prague	162 00
6	Pablo	Rodriguez	Vratislavova	Prague	128 00

	name	surname	1	3	4	6
1	Mary	Smith	0	0	1	1
3	Anne	Maxwell	0	0	1	0
4	John	Newlin	0	0	0	0
6	Pablo	Rodriguez	0	1	0	0

	cart
1	product : T1, quantity : 2, product : B4, quantity : 1
3	product : H1, quantity : 1
6	product : B3, quantity : 2

Figure 0.10: An example of TDM

U-Schema

Recently, the approach [30] integrating schemas of distinct logical models, namely relational, graph, key/value, document and columnar models, was proposed. The proposal also includes a mapping between the respective logical model schema and the integrating schema and vice versa.

The basic structural constructs of the proposed approach are as follows:

- Model U-Schema $U = \{s_1, \ldots, s_n\} \subseteq S$, $n \in \mathbb{N}$ is a set of SchemaTypes $s \in S$.

- The set of SchemaTypes S is the union of EntityTypes ($E \subseteq S$) representing classes of real-world objects and RelationshipTypes ($R \subseteq S$) representing relationships between them. Internally, EntityType $e \in E$ is represented as a tuple $(n_s, root, V_s)$ and RelationshipType $r \in R$ is a tuple (n_s, V_s), i.e., each SchemaType is assigned a name n_s and contains a subset of StructuralVariations $V_s \subseteq V$. In addition, EntityType e contains a Boolean feature $root$ indicating whether it is a standalone or nested entity type. Hence, only the EntityType can form a hierarchical structure.

- Each StructuralVariation $v \in V$ is represented as a tuple (id, F_v, $count$, $firstTS$, $lastTS$) such that id is an integer identifier, $F_v \subseteq F$ is a subset of the properties, $count$ is a feature capturing the number of existing instances of a particular StructuralVariation, and the timestamps $firstTS$ and $lastTS$ store the time of creation of the first and last instance of that variation.

- The set of properties F is composed of LogicalFeatures (i.e., keys $KEY \subseteq F$ and references $REF \subseteq F$) and StructuralFeatures (i.e., attributes $ATT \subseteq F$, and aggregates $AGG \subseteq F$) such that:

- The attribute $att \in ATT$ can be primitive (e.g., Number, String, Boolean, JSON,[21] BLOB), collections (e.g., list, tuple, set, map), or the special type Null. Each attribute att is modeled as a tuple (n_{att}, t_{att}, opt_{att}, key_{att}, REF_{att}, is_{att}) such that n_{att} assigns a name to the attribute, t_{att} is the data type of the attribute, opt_{att} specifies the optionality of the attribute, $key_{att} \in KEY$ is a reference to the (none or only) key of which the attribute may be a part, $REF_{att} \subseteq REF$ is a subset of the references of which the attribute is a part, and the is_{att} attribute models specific behaviour based on the attribute type.
- The key $key \in KEY$ is modelled as a tuple (n_{key}, A_{key}) such that n_{key} is the name of the key and $A_{key} \subseteq ATT$ is a subset of attributes that constitute the key.[22]
- The reference $ref \in REF$ is modelled as a tuple (n_{ref}, A_{ref}, $refsTo$, $lBound$, $uBound$) such that n_{ref} is the name of the reference, $A_{ref} \subseteq ATT$ is a subset of the referencing attributes,[23] $refsTo \in E$ is the referenced EntityType, and $lBound$, $rBound$ are the lower and upper cardinality bounds, respectively.
- The aggregation $agg \in AGG$ is modeled as a tuple (n_{agg}, opt_{agg}, $lBound$, $uBound$, V_{agg}) such that n_{agg} assigns a name to the aggregation, opt_{agg} determines the optionality of the aggregation, $lBound$, $rBound$ are the lower and upper cardinality bounds, and $V_{agg} \subseteq V$ is the set of structural variations that are aggregated (nested).

Example 0.13. Figure 0.11 (a) illustrates the U-Schema of the graph data and Figure 0.11 (b) the U-Schema of the document data from the Example 0.1. Note that although edges are represented internally in the graph as a pair of properties from, to, U-Schema represents an edge as a RelationshipType without any properties, and the connection between two customers (otherwise realised by an edge) is represented by a reference at Customer. In the case of the U-Schema document model, note that only the EntityType Order is root, reflecting the fact that all other EntityTypes are nested. □

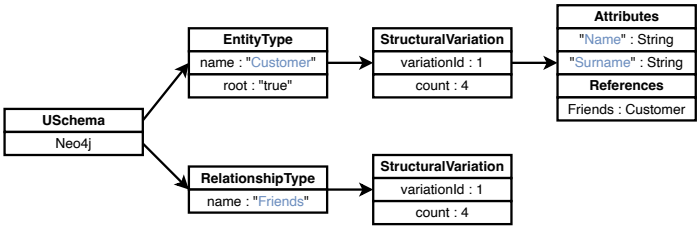
U-Schema allows to define two variants of the model: (1) in the so-called *full variability* all structural variations of all EntityTypes and RelationshipTypes are stored, while (2) in the so-called *union schema* there is only one structural variation for each SchemaType. Note that the conversion from the *full variability* to the *union schema* version is a lossy conversion, hence the reverse conversion is not possible.

Furthermore, the authors introduce so-called forwards mapping (i.e., a mapping of the logical model schema to the U-Schema) and reverse mapping (i.e., a mapping in the opposite direction). In the former case, there is a natural correspondence between each element of the logical schema and an element of the U-Schema. However, in the latter case, the U-Schema may contain elements that

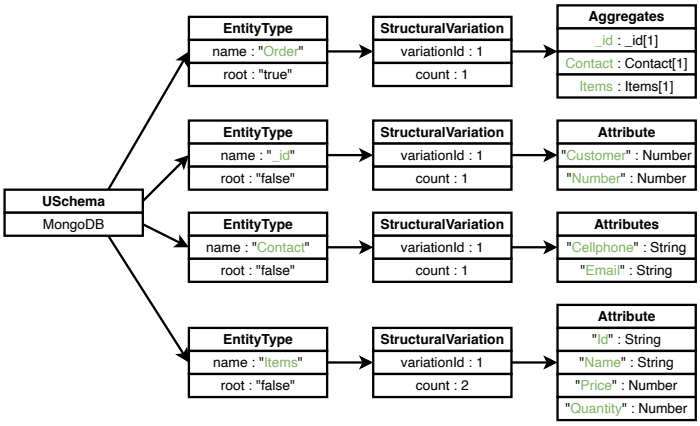
[21] Note that the authors consider the nested data model represented by JSON or JSONB type in PostgreSQL as a black box and not as a structure.

[22] Note that U-Schema allows primitive attributes as part of the key, as well as collections and the special type Null [30].

[23] Note that, similar to the key, references can be collections and the special value Null in U-Schema.



(a)



(b)

Figure 0.11: An example of U-Schema for graph (a) and document model (b)

are not present in a particular logical model, e.g., the relational model does not contain structural variations,[24] the graph model does not support aggregates, and conversely, most logical models do not contain the RelationshipType. Hence, the unification of data models is limited, as model-specific constructs are introduced into U-Schema, which also makes it difficult to extend the approach to support other logical models if needed.

Finally, the authors proposed the language Athena [60], which allows the definition of logical schemas as U-Schemas.

Comparative Summary

Table 0.3 summarises the features of selected abstract models at the logical layer. All the observed approaches allow us to represent data, however, the schema is approached in various ways. Most approaches can be considered schema-full, however, NoAM approach lacks an explicit schema, thus representing a schema-less approach. The abstraction of logical data models varies. Typically, if the abstract model is based on the graph (or category), it allows to represent relational or graph data (i.e., aggregate-ignorant models). Conversely, if the model is an array-like or aggregate-like, mostly aggregate-oriented models are supported. However, multi-model data is only considered in the U-Schema approach, and

[24]Note that the U-Schema treats missing value differently. While in the relational model, missing value leads to an optional property within one structural variation, in the graph model, missing value leads to two different structural variations, and in the document model we further distinguish a variation with the special type Null.

Table 0.3: Comparison of logical layer modeling approaches

	CGOOD [19]	Spivak et al. [20]	APG [21]	NoAM [27]	AA [28]	TDM [29]	U-Schema [30]	MM-cat [3]
Data structure	Typegraph	Category	Category	Aggregate	Matrix	Tensor	Graph	Category
Schema	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
Data	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
Relational model	Partial	Partial	Partial	No	Partial	Partial	Partial	Yes
Array model	No	No	No	No	No	No	No	Yes
Graph model	No	No	Yes	No	Yes	Yes	Yes	Yes
RDF model	No	Yes	No	No	No	No	No	Yes
Key/value model	No	No	No	Yes	Yes	Yes	Yes	Yes
Document model	No	No	No	Yes	Yes	No	Yes	Yes
Columnar model	No	No	No	Yes	Yes	No	Yes	Yes
Multi-model data	No	No	No	No	No	No	No	Yes
Complex ID	No	No	No	Yes	Yes	No	Yes	Yes
Multiple IDs	No	No	No	No	No	No	No	Overlapping
References	Implicit	Intra	Implicit	No	No	No	Yes	Intra, Inter
Complex IC	No	∼	∼	No	No	No	No	No
Complex types	Structure, Array	No	Array	Structure, Array	Implicit	No	Structure, Array, Tuple, Set, Map	Structure, Array, Tuple, Set, Map
Union type	No	No	Yes	Implicit	No	No	Yes	Yes
Ordering	No	No	No	Yes	Yes	No	No	Yes
Data redundancy	No	No	No	No	No	No	No	Yes
Querying	Graph patterns	CQL	CQL	No	Matrix algebra	Tensor algebra	No	No
Data Migration	Intra	Yes	Yes	No	No	Yes	Yes	Yes
Evolution management	Yes	Partial	Partial	No	No	Partial	Yes	Yes

only in a limited way. The approach treats multi-model data as a set of disjunctive models, i.e., it does not consider features arising from their connections, e.g., inter-model references, redundancy (see Paper III) and (in)consistency [61]. ★

Nevertheless, even in the case of support for individual data models, approaches are often limited to a minimal set of constructs, e.g., simple identifiers, simple properties and basic data structures (i.e., a subset of complex properties). Only approaches NoAM and AA allow to model complex identifiers, while completely missing, e.g., the possibility to specify multiple identifiers and thus to represent overlapping identifiers typical, e.g., for a relational model, hence multi-model data. In the case of references, the approaches are limited to (implicit) intra-model references only.

Finally, approaches often introduce minimal sets of operations that can be composed to perform more complex operations that express, e.g., querying, data migration and some evolution management operations.

0.3.3 Open Questions and Challenges in Data Modelling

Indeed, existing data modelling approaches seem promising for representation of multi-model data at the conceptual and logical layers. However, none of these approaches allows us to represent all the features of multi-model and underlying data in their natural form. Hence, there remain open questions that need to be addressed.

Conceptual Layer

Naturally, approaches that model the conceptual layer should support multi-model data, as they intentionally hide the specific properties of platform-specific (logical) representations. However, traditional approaches, such as ER and UML, are primarily closely associated with the relational model (i.e., normalised data) and do not necessarily reflect the properties of additional models (e.g., denormalised data). In other words, the translation of the conceptual schema is not straightforward and is often ambiguous in the case of a combination of models.

The following **challenges C1 − C9** need to be discussed accordingly in order to design an extended approach that fully represents multi-model data at the conceptual level:

C1: *Elements of the conceptual layer.* The question is whether we need to distinguish classes of objects from relationships and properties at the conceptual level. In practice, conceptually equivalent constructs are often represented differently at the logical level, and different conceptual constructs are represented in the same way. This is particularly obvious in aggregate-oriented models where, e.g., a structured attribute is interchangeable with a combination of a class of parent object, a class of (nested) objects, and respective relationships.

C2: *Updated concept of a property.* The traditional concept of a property can be understood as a pair $(name, value)$, where *name* is statically bound to *value* (thus easily representable at the conceptual level). However, with the advent of data models that allow data to be represented by a property

of type map, the static binding is replaced by a dynamic one, i.e. *name* becomes a part of the data. This trait, i.e., the dynamic naming of a (nested) property, would also be useful to represent at the conceptual layer.

C3: *Required explicit/implicit identifier.* We believe that the motivation for the introduction of a required and explicit identifier for classes of objects at the conceptual level was, among other things, to enable the unambiguous connection of different instances of classes, i.e., to realise relationships between them, whereby a particular instance of a relationship is implicitly identified just by the participants. Moreover, properties can be seen as trivial object classes that contain only an identifier and thus explicitly identify themselves [20]. Hence, it is possible to identify all the elements of conceptual model, which could allow a further level of unification.

C4: *Ordering of properties.* Currently, the conceptual level only allows to represent property names within a single class, but their relative order cannot be captured.

C5: *Structured property.* Arguably, given the properties of the relational model, a structured property, e.g., in ER, can only have trivial depth and allows only trivial cardinality (i.e. one-to-one). However, in practice we also encounter unrestricted structured properties that have non-trivial depth and where sub-properties are repeated. Traditionally, such a state can be represented by a combination of object classes and relations, but this goes against the concept of conceptual modelling, as it is a structured property and not a combination of classes and relationships.

C6: *Multivalued identifier.* Considering simple (single-property) identifiers, currently only properties with trivial cardinality (i.e., one-to-one) can be used to identify an instance within an object class.

C7: *Identifier of a weak class type.* The identifier of the weak class type is formed by the identifiers of all types involved in the relationship. In some cases, it may be sufficient for the weak identifier to consist of only a subset of the identifiers of the types participating in the relationship.

C8: *Complex integrity constraints.* Ideally, we need to capture integrity constraints (even complex ones) at the conceptual layer, as integrity constraints are a natural and often overlooked part of the schema. In particular, ER is limited to the representation of identifiers, whereas UML using OCL allows the description of multiple types of integrity constraints including, e.g., business rules.

C9: *Aliasing properties.* Each element of the conceptual schema is assigned only with a single name. However, in certain cases it is useful to introduce synonyms, e.g., properties `father` and `mother` could be referred to as property `parent`. Consequently, the special case of inheritance, though at the property level, could, e.g., simplify (conceptual) querying.

Logical Layer

Based on our findings, a fundamental construct of popular logical data models is a property, i.e., the mapping $p : N \to V$, where $n \in N$ is the name and $v \in V$ is the value. Yet the models differ, e.g., in the way the properties are aggregated into larger logical units (e.g., a flat table or a hierarchical tree structure), in the enforcement of the order of properties (i.e., order-oriented/ignorant models), the possibilities of identifying larger units, the support for references between larger units, the way missing data is represented, or the attachment of structural information to the data (i.e., structured, semi-structured, unstructured formats). A common underlying construct (i.e., name-value pair) can be used to design a unified (abstract, not necessarily logical) model. In designing such a model, the following **challenges L1 – L7** must be properly tackled:

L1: *Unified logical model.* A question is whether we need a unified layer of multi-model data at the logical level in the sense of a single data model (more or less painfully combining the properties of the models) or rather an abstract model that merely overlays the existing logical models. In the latter case, we could continue to take advantage of features of existing approaches, e.g., representing data as a graph if we query primarily over relationships, or aggregates if we repeatedly call queries aggregating data, etc., while treating all models uniformly.

L2: *Representation of model-specific constructs.* The unifying layer should allow for uniform capturing of semantically similar constructs across different models (e.g., a nested JSON document as a map or a tuple as an array) and also capture model-specific constructs (e.g., complex and overlapping identifiers).

L3: *Missing data.* Logical models differ in the way they represent missing data. For instance, the relational model represents missing data as a null (meta)value, the graph model (e.g., property labelled graph implemented in Neo4j) as non-existent properties, while the document model allows combining both approaches. The question is whether it is possible to represent missing data in a unified abstract way.

L4: *Inter-model links and references.* Logical models support various forms of intra-model references (e.g., traditional references and embedding). However, references and embedding across different models are a natural feature of multi-model data. For example, PostgreSQL allows embedding of XML and JSON document data into a relational table by introducing a special column type.[25] Hence, we need an abstract model that considers both intra- and inter-model references.

L5: *Conceptual to logical layer mapping.* Ideally, we need a unified algorithm that allows mapping between conceptual and logical layers, regardless of the specific properties of the logical layer.

[25] https://www.postgresql.org/docs/current/

L6: *Propagation of changes.* The unified data model should be designed with a perspective of a uniform way of propagating changes in different (underlying) logical models, but also across these models. The propagation of schema and data changes should also be accompanied with the propagation of query changes.

L7: *Extensibility* towards new data models that currently does not exists but are based on similar idea, e.g., where the most general construct is a pair $(name, value)$.

0.3.4 Contribution: Framework MM-cat

So far, we have discussed existing approaches and challenges that need to be addressed in order to appropriately represent multi-model data at the conceptual and logical level. Now, we turn to a commentary of the actual solution and its incremental extension to meet additional requirements, i.e., not just representing multi-model data in a unified way, but enabling data migration, querying, and evolution management. In this subsection, we mainly discuss data representation, while migration and evolution management are addressed in separate Section 0.5 and querying forms our future work.

We have first analysed selected existing data representation solutions at the conceptual level [31, 32, 18] and logical level [19, 20, 21, 27, 28, 29], as well as other approaches [48, 62] describing, e.g., integrity constraints, and we have verified their applicability to multi-model data [26], thereby also revealing drawbacks and open questions. Being aware of the limitations of the solutions, we have ★ outlined the concept of a unified schema representation [63] and the vision of a comprehensive framework built on top of solid formal foundations and allowing ★ the management of multi-model data in a unified yet natural way (see Paper I).

As mentioned above, the development of our approach has been gradual. In the beginning, we mainly considered schema and data representation at the conceptual level. The goal of developing an early concept was to test whether we could apply category theory at all to the representation of multi-model data. In the second stage, we added support for logical-level mappings and introduced the notion of mapping between logical and conceptual levels. In addition, we used this mapping in the design of algorithms implementing data migration. Finally, we have extended the approach to naturally support complex schema and data change operations, i.e., to enable evolution management.

Early / Original Concept

In the early stages of designing the unifying conceptual model, we tried to represent multi-model data as simply as possible, i.e., using only the elementary constructs of category theory (e.g., categories, functors, and their composition). The motivation for this decision was the desire to create a model that has the potential for broad extension and application, and thus it is appropriate to avoid complex constructs at this elementary level. Our inspiration came from the works [18, 19, 20] modelling conceptual, object-relational, or relational schemas, as well as vision [56], in which the authors outlined an extension of the [20] approach towards supporting additional data models.

Our approach (see Paper II) is based on definition of a schema category, ★ an instance category, and their mutual mapping. The *schema category* $\mathbf{S} = (\mathcal{O}_\mathbf{S}, \mathcal{M}_\mathbf{S}, \circ_\mathbf{S}, 1)$ captures the data structure and describes the basic integrity constraints. Objects in $\mathcal{O}_\mathbf{S}$ represent the classes of real-world objects, relationships between them, and properties of both object and relationships (challenge C1) in a uniform way. Although the user may still distinguish between the three object types (i.e., object, relationship, and property), this is not necessary from the categorical point of view. Additionally, an object is internally represented as a pair (*superid*, *ids*), with *superid* representing a concept of a superidentifier and *ids* being a set of identifiers. Such representation allows us explicitly include the identifier also for relationships and properties, if appropriate (challenge C3). Morphisms in $\mathcal{M}_\mathbf{S}$ model the concept of object relations. Internally, a morphism is represented as a pair (min, max), which allows us to model the traditional concept of cardinalities, while at this stage we only consider trivial cardinalities having $min \in \{0, 1\}$, $max \in \{1, *\}$, $min \leq max$. The composition of morphisms $\circ_\mathbf{S}$ also reflects the composition of cardinalities. For convenience, the user can also still distinguish morphisms based on the type of linked objects into property, relation, and ISA hierarchy morphisms, but from a categorical point of view this is again not essential.

Instance category $\mathbf{I} = (\mathcal{O}_\mathbf{I}, \mathcal{M}_\mathbf{I}, \circ_\mathbf{I}, 1)$ represents data in a unified way. This category structurally corresponds to the schema category, i.e., there is a functor $F : \mathbf{I} \to \mathbf{S}$ that assigns a schema to the data (as inspired by the approach [19]). Alternatively, we can represent the instance category as a category of sets and functions, i.e., we can build it on top of category **Set** (this variant is described in ★ Paper II) or **PartSet** allowing to additionally represent missing data (as inspired by the approach [20]). In contrast, if we were to represent schema and data directly using graph theory or set theory instead of category theory, the choice of instance representation would not be so straightforward.

Finally, we propose an algorithm to translate the ER schema into the corresponding schema category. In other words, the expressive power of our approach is at least comparable to the ER model. In fact, our approach even extends ER as it represents properties of other logical models (i.e., not only relational, but also array, graph, RDF, key/value, document, and column) at the conceptual level (challenge C2, challenge C5).

Extension towards Data Migration

In the course of proposing the data migration algorithms, we decided to extend the internal representation of objects and morphisms of the schema category and added dual (not necessary inverse) morphisms to the so-called *property morphisms*. The internal representation of schema category objects has been extended to $(key, label, superid, ids)$, where elements *key* and *label* have been added. The former is represented as $key \in \mathbb{N}$ and allows unambiguous identification of the object $O \in \mathcal{O}_\mathbf{S}$. The latter allows the object to be assigned a name, which, unlike the previous proposal, now need not be globally unique.

Morphisms are modelled internally as $(signature, dom, cod, min, max)$. Compared to the previous design, *signature*, *dom* and *cod* are added. The *signature* assigns ϵ to identity morphisms, $n \in \mathbb{N}$ to base morphisms, and concatenation of *signature*s of morphisms being composed separated by '.' (dot) to compound

morphisms. The pair *dom*, *cod* represents the domain and codomain of the morphism. The new features allow morphism identification as follows: (1) an identity morphism having $signature := \epsilon$ is determined by the pair dom, cod, since each object can have just one identity morphism, and (2) a non-identity morphism is identifiable only by *signature*, as this value is unique for non-identity morphisms. Note that the $\circ_{\mathbf{S}}$ is also modified to reflect the composition of *signature*, *dom*, and *cod*.

The previous definition of the schema (and instance) category considered dual morphisms only between pairs of objects $O_1, O_2 \in \mathcal{O}_{\mathbf{S}}$ corresponding to classes of real-world objects and their relationships. The new definition of schema category considers dual morphisms even when O_1 or O_2 correspond to a property object. The reason for this change is to allow arbitrary (directed graph) traversal of the schema category, which we exploit during data migration and it is also a preparation for querying based on pattern matching.

In addition, we introduce a concept of *mapping between logical and conceptual layers*, specified for each kind κ by $\mathfrak{M}_\kappa := (\mathcal{D}, name_\kappa, root_\kappa, morph_\kappa, pkey_\kappa, ref_\kappa, P_\kappa)$, where $\mathcal{D}$ denotes a particular database component, $name_\kappa$ represents the name of the kind κ, $root_\kappa$ or $morph_\kappa$ refers to the root property of κ, $pkey_\kappa$ describes the structure of its identifier, ref_κ captures the references from κ to κ' (challenge L4), and P_κ captures the hierarchical structure along with the model-★ specific constructs of the logical representation of kind κ (challenge L2) (see Paper III for more details). Hence, the mapping allows us to decompose the schema category into logical units corresponding to each kind κ (challenge L5, challenge L1). Note that by the addition of the mapping of the logical to the conceptual layer, the approach is also ready to extend the support for additional data models (challenge L7).

The mapping, together with the extended schema category, forms the foundation of universal algorithms of data transformation from logical to categorical representation and vice versa. Moreover, these algorithms allow data migration from any input to any output combination of logical models, i.e. the multi-model to multi-model migration. Finally, also a uniform approach to missing data, even though they may be represented differently at the logical level, is supported (challenge L3).

A complete solution is implemented as the open-source academic prototype ★ *MM-cat* [64]. This tool allows a multi-model schema to be represented by a graph freely generating a schema category (see Definition 5). In addition, the tool allows the schema to be automatically extracted from a conceptual model (e.g., ER and UML). Naturally, the tool provides a unified approach to logical models (including their specific properties) by mapping the logical model to a schema category. Finally, *MM-cat* implements transformation algorithms implementing data migration. For proof-of-concept purposes, the prototype supports PostgreSQL (relational and document model, i.e., multi-model representative), MongoDB (document model representative) and Neo4j (graph model representative). The specific properties of the database systems are implemented using ★ so-called wrappers (see Paper III for more details).

0.4 Schema Inference

An attractive feature of a majority of NoSQL databases is the possibility of storing data in kinds without a previously defined schema. From the user's perspective, this is a simple and flexible way of storing data. However, in various use cases we still require the knowledge of an explicit schema, e.g., in the case of querying, migration, and data evolution. In such cases, it is necessary to infer the non-existing schema secondarily, from the already stored data.

Currently, there are a number of approaches for schema inference over a single data model, mostly for aggregate-oriented database systems based on JSON and XML document models. However, these approaches often suffer from various drawbacks, e.g., the inferred entities contain too many (optional) properties (i.e., the schema is not very clear to users), the inferred schema does not respect the order of the properties (e.g., approaches that infer the schema of JSON documents rarely consider the order of the elements in a JSON array), or they infer integrity constraints in a very limited way or not at all. Moreover, the inferred schemas are often complicated also from the data modelling perspective. For example, the UML [32] does not allow to model an inferred schema if it contains a union type. Therefore, also new schema description formats have emerged, though often non-standardised, tailored to a single particular system, or supported only by a particular schema inference approach (e.g., Baazizi's proprietary language [65]), making it difficult to compare schemas inferred by different approaches or even over different database systems. Hence, we aim to thoroughly analyse the problem of schema inference over multi-model data and verify whether it is possible to extend any of the existing approaches in order to infer a schema for multi-model data, or if a completely new approach is necessary.

In this section, we first describe the state-of-the-art approaches to schema inference over individual data models. We then elaborate on five recent JSON schema inference approaches and compare them statically to answer the question of whether any of the selected approaches can be used to infer a schema of multi-model data. Based on the analysis of the selected schema inference approaches, we discuss a set of open questions and issues in schema inference in the context of multi-model data. Finally, we present our fully multi-model schema inference approach.

0.4.1 State of the art

The research on inferring the implicit schema of data is not new. It includes not only modern single-model NoSQL databases, but also older technologies such as XML and RDF [66]. The principles of the different algorithms are similar, differing only in the features supported by each format, e.g., capturing the order of properties in case of XML. In addition, the approaches are often scalable and support parallel processing of Big Data.

Graph and Linked Data Schema Inference There are several approaches dealing with schema inference in the graph model and Linked Data. Lbath et al. [67] focus on inferring simple and complex types in a property labelled graph, including its hierarchy and cardinality of edges. Galinucci et al. [68] address

schema inference for RDF documents, where they identify aggregated hierarchies and repeating patterns in Linked Open Data. Finally, Bouhamoun et al. [69] tackle the horizontal scaling problem of processing large amounts of RDF data and present a method based on pattern extraction in linked data.

Key/Value Schema Inference The problem of inferring schema in key/value stores and then transforming the data into (flat) relational tables is studied by DiScala and Abadi [70]. In this case, the authors work with systems that store structured JSON documents in place of values, but treat them as black boxes at the database system level.

XML Schema Inference There are several comprehensive papers dealing with approaches to inferring schemas in XML documents [71]. A comparison of existing heuristically based approaches, including open problems, is provided in [72]. These are primarily older approaches, popular before the advent of a more popular format – JSON. A comparison of the grammar inferring approaches can then be found in [73].

Heuristic approaches [74, 75, 76, 77, 78, 79] are based on generalising the schemas of individual XML documents based on a set of predefined heuristic rules. These methods can be further subdivided according to the chosen strategy: (1) The approaches [75, 76, 78] gradually generalise the schema until they reach the desired solution. (2) The approach [77] generates a large number of candidates and selects the most suitable schema. (3) Algorithms called *merging state* [76, 78] are based on searching a heuristically selected subspace of all possible schema generalisations of a given XML document. They represent the schemas as states of a prefix tree automaton and construct sub-optimal solutions by merging its states.

Alternatively, there are methods based on *grammar inference* [80, 81, 82, 83, 84, 85, 86]. These methods consider an XML schema as a grammar and the XML document corresponding to the schema is the word generated by the grammar. Moreover, this problem can be reduced to the extraction of a set of regular expressions, where one regular expression describes one XML element. Moreover, the approaches exploit additional information besides the XML documents, e.g., a predefined maximum number of nodes of the target automaton, since according to Gold's theorem [87], regular languages cannot be identified based on positive examples alone.

The majority of the existing approaches represent the resulting schema using the DTD language. Only the approaches [86, 78, 88] represent the schema using the XML Schema language.

JSON Schema Inference Although the JSON and XML document formats are very similar (i.e., both are semi-structured hierarchical data formats), schema inference approaches for XML documents are often not applicable to large collections of JSON documents [16], not only because of the differences between these formats (e.g., XML is order-preserving and duplicate-allowing while JSON is order-ignorant and duplicate-prohibiting), but also because the existing schema inference approaches for XML documents do not assume large data collections (i.e., Big Data) and their scalability and parallelisability are thus limited.

A comparison of the static properties of selected JSON schema inference approaches is addressed in works [89, 90]. At the same time, popular approaches are also described in our work [16], where we additionally investigate the applicability of selected approaches for schema inference over multi-model data. Finally, the comparison of dynamic properties is addressed in our paper [91]. ★ ★

Sevilla Ruiz et al. [92] propose an approach of inferring versioned schemas from document-based NoSQL databases. The foundation of the approach is an abstract model based on the *Model-Driven Engineering* (MDE). This work is followed up by Chillon et al. [93], who address the visualisation of NoSQL database schemas and propose extensions needed to visualise aggregate-oriented data. Most recently, Fernandez et al. extend the abstract model by adding the support for relational and graph data [30].

Klettke et al. present a complex solution for managing NoSQL schemas [94], including an approach for reconstructing schema evolution history in so-called *data lakes* [95]. This research is followed by a tool *jHound* [96] that enables profiling of JSON data, e.g., searching for structural outliers. Finally, the tool *Josch* [97] allows schema extraction from JSON data, schema refactoring, and subsequent validation against the original dataset.

Baazizi et al. [65] propose a horizontally scalable approach for parameterised schema inference from large collections of JSON documents. They also introduce a custom and compact language for describing the resulting JSON schema.

Izquierdo and Cabot [98] proposes an approach to infer schemas for web services based on JSON documents. The authors also provide a web application along with a visualisation tool [99].

An approach of inference of schema over collections of JSON documents is also presented by Frozza et al. [89]. In contrast to previous works, the authors consider inference of data types specified by the BSON standard.[26] Unfortunately, their approach has limited parallelisability.

Last but not least, Wang et al. [100] propose a schema inference method over document repositories based on finding equivalent subtrees (i.e., frequently repeated hierarchical structures).

The JSON Schema language is primarily used to describe the inferred schema of JSON documents. The formal model of this language is discussed by Pezoa et al. [101], while the general description of the JSON type system is discussed by Baazizi et al. [102].

Columnar Schema Inference Frozza et al. have also proposed an approach for schema inference over columnar NoSQL databases [103], specifically supporting the inference of implicit schema from the *HBase*.[27]

Summary There is a number of approaches aimed at inferring a schema over a particular data model. Unfortunately, to the best of our knowledge, there is currently no approach applicable to infer a schema over a combination of data models, i.e., multi-model data. At first sight, it may seem that existing single-model approaches can be also applied to infer a multi-model schema, but as we will show in the next subsection, this idea is not feasible in practice, as the individual

[26] https://bsonspec.org
[27] http://hbase.apache.org

data models often have contradictory features. Moreover, the combination of models itself is a complication, as we must additionally consider references and redundancy across models, not just within a single data model.

0.4.2 Closely Related Single-Model Approaches

The utilisation and extension of verified single-model approaches seems promising for the inference of a multi-model schema. To validate this idea, we chose inference approaches over JSON documents because the JSON format is sufficiently complex to cover a variety of data constructs, at least at first sight. Moreover, these approaches are often horizontally scalable and thus covering also high volumes of data.

In particular, we focus on five selected schema inference approaches, namely:

1. The approach proposed by Sevilla Ruiz et al. [92] working with the concept of distinct versions of entities.

2. The approach of Klettke et al. [94] using a graph structure to represent a schema and able to detect outliers.

3. The approach of Baazizi et al. [65] which introduces a comprehensive and massively parallelisable method for inferring of schemas.

4. Izquierdo and Cabot [98] approach which can infer a schema from multiple document collections.

5. The approach of Frozza et al. [89] which is able to infer schemas including data types as introduced in BSON.

We compare the approaches statically, i.e., we focus mainly on their basic principles and algorithm scalability, input and output parameters, possible support for structural components beyond the JSON format, distinguishing between optional and required properties, support for inference of integrity constraints, and detection of redundancy in the data. We also verify whether the selected approaches are applicable to infer a multi-model schema. Table 0.4 summarises the comparison of key characteristics of the selected approaches. (A comparison with our approach is also included; however, our approach is not discussed in the following paragraphs. It is introduced in Paper IV.)

Inference Process The majority of approaches generate a schema based on all documents in the input collection. An exception is the approach of Izquierdo and Cabot which retrieves documents from web services. Moreover, the approaches of Sevilla Ruiz et al. and Frozza et al. minimise the input document collection into a collection that contains only structurally distinct documents. A common feature of all approaches is the replacement of property values by the names of the supported primitive data types they encounter.

Scalability Design Based on the theoretical design, most approaches are horizontally scalable. Unfortunately, their implementations are usually not parallelised. Sevilla Ruiz et al. use MapReduce in order to select structurally distinct documents from the input collection. Unfortunately, in the worst case, where all documents are structurally distinct, the scalability of this approach is limited since the subsequent processing of the minimal collection of structurally distinct documents (i.e., in this particular worst case, the entire input document collection) is not parallelised. In contrast, the approach of Baazizi et al. is fully horizontally scalable because Apache Spark is used throughout the schema inference process.

Implementation The approaches of Sevilla Ruiz et al. and Izquierdo and Cabot are Java applications running in the platform-independent Eclipse environment. Klettke et al. implement their approach as a Spring Boot application (i.e., as a platform-independent application built in Java) and additionally allow the approach to be deployed as a Docker container. Baazizi et al. implement the approach using Scala as an Apache Spark job. Finally, Frozza et al. approach is implemented as a javascript web application.

Input All approaches support schema inference from a collection of JSON documents. Frozza et al. approach also supports BSON data. In addition, the approaches of Sevilla Ruiz et al. and Klettke et al. allow arbitrary aggregate-oriented data to be converted to JSON data, and thus apply their approach for schema inference over these models as well. However, by converting data from other models to JSON documents, we may lose structural information, e.g., because JSON does not preserve the order of features and does not allow explicit representation of complex data structures such as maps, sets, and tuples. Hence, a schema inferred this way may not be accurate.

Moreover, only the approaches of Sevilla Ruiz et al. and Izquierdo and Cabot allow the inference of a schema from the entire input database, i.e., from a set of collections. The remaining algorithms infer schemas only over individual collections.

Output The approaches represent the inferred schema by means of textual or graphical languages. Klettke et al. and Frozza et al. approaches use the JSON Schema language to describe the schema, although they differ in the details of use. Baazizi et al. represent the schema with their own compact but complex proprietary language. Finally, Sevilla Ruiz et al. and Izquierdo and Cabot represent the inferred schema as a UML class diagram.

Structural Components The selected approaches differ in the extent to which they support the inference of different structural components of JSON documents. Most approaches are only able to infer a basic set of primitive types (i.e., *String*, *Number*, and *Boolean*) and some complex types (i.e., nested objects and arrays). Whereas, of the approaches we observed, only Frozza et al. allows the inference of the extended data types introduced in BSON. Unfortunately, the approaches do not allow distinguishing other complex data structures, namely maps, sets, and tuples, as even the JSON format does not distinguish these structures from nested

structured objects (also representing, e.g., maps) and arrays (also representing, e.g., sets and tuples).

Optional Properties The majority of approaches distinguish between required and optional properties, differing only in the way they are detected. The approaches of Klettke et al. and Frozza et al. compute differences in the occurrence of individual properties compared to the occurrence of parent properties. If the parent property occurs more frequently, the child is marked as optional. Sevilla Ruiz et al. are able to detect optional properties using set operations over entity versions (intersection of properties of two or more entity versions returns required properties, while union of entity versions minus their intersection returns a list of optional properties). Baazizi et al. detect optional properties during the merging of two document schemas. Finally, Izquierdo and Cabot is the only approach unable to distinguish between optional and required properties. In this case, we consider all properties as required.

Union Type The majority of the compared approaches work with the concept of union type. The approaches of Klettke et al. and Frozza et al. use the JSON schema keyword *oneOf*, while Baazizi et al. represent a union type as a concatenation of types with the "+" character. Only the approaches of Sevilla Ruiz et al. and Izquierdo and Cabot do not consider the concept of union types. The former consider versions of entities where union types cannot naturally occur. As for the latter, the type of a property is expressed only by the most general of the identified types, e.g. as *String*.

Order Preserving All approaches treat JSON documents as a set of unordered properties and therefore do not consider order detection. Unfortunately, the order is not even considered in the case of array elements, where the order matters.

Integrity Constraints Integrity constraints are detected only to a limited extent or not at all. Identifiers (whether simple or complex) are not detected by any of the approaches. References are only partially detected in Sevilla Ruiz et al. approach, based on a naming convention. The inference of other (complex) integrity constraints is not considered at all, e.g., ranges of values of individual properties or mutual dependency between values of different properties.

Data Redundancy Although redundancy in data is a common feature of NoSQL document stores, most of these approaches do not detect this feature. Only the approach of Izquierdo and Cabot allows merging the schema of two collections (i.e., considering them as redundant) if they have identically named properties, but does not verify this fact at the data level.

Additional Features To conclude the comparison, let us also focus on a couple of specific features supported by just some of the covered approaches. In particular, the approach by Sevilla Ruiz et al. allows for the visualisation of the inferred schema [93], including the visualisation of entities and relationships using UML. The approach by Klettke et al. allows us to analyse documents using a

proprietary profiling tool *jHound* [96]. The approach also detects errors and outliers, i.e., additional or missing properties. To continue, the approach by Baazizi et al. is suitable for inferring schemas over large JSON document collections due to the massive parallelisation applied. Fusion properties, i.e., associativity and commutativity, enable the evolution of an already inferred schema, making this approach suitable for collections of documents that are rapidly evolving. The approach by Izquierdo and Cabot is designed to generate schemas for sets of web services producing JSON documents with similar features. With a global schema, the user then gets an idea of which specific services to call to get the requested information. A part of this approach is also a schema visualisation tool [99], which produces a schema compliant to JSON Schema. Finally, the approach by Frozza et al. is designed to work over the *MongoDB* database and supports the extended BSON format.

As illustrated in Table 0.4, the approaches do not consider natural features of multi-model data, e.g., variety of data types, complex structures, ordering of properties and theirs duplicities, identifiers, inter- and intra- model references and redundancy, and complex integrity constraints in general. Thus, despite the complexity of JSON model, the strategy of converting multi-model data into JSON data and inferring its schema is not sufficient as we lose critical schema information.

0.4.3 Open Questions and Challenges in Schema Inference

We believe that in order to be able to infer schema even for multi-model data, we need to address the limitations of existing approaches and extend them appropriately. Therefore, we provide the following list of **challenges I1 – I10** in the area of schema inference for multi-model data.

I1: *Integrity constraints.* Generally, the schema not only describes the structure of the data, but it may include a list of integrity constraints, e.g., identifiers, references, ranges of property values, and rules describing complex dependencies between properties. At first sight, this challenge goes beyond the bounds of some data models and the languages describing their schema. For instance, JSON and JSON Schema do not allow modelling complex integrity constraints. However, this does not mean that we do not have implicit integrity constraints in JSON data. In the case of multi-model data, we could use, e.g., OCL [48] to describe integrity constraints if the underlying data model does not explicitly support them.

I2: *Values of properties.* Most of the observed approaches infer the schema only from the structure of the data, i.e., the names of the features and mutual hierarchy. The values are only converted to their data types and further processing of the values themselves is omitted. We believe that, e.g., a statistical analysis of property values can refine the inferred schema, e.g., allowing the inference of identifiers, references, and some other integrity constraints.

I3: *Multiple data models.* A multi-model schema is a union of schemas from data represented by various logical models. Underlying data models may

Table 0.4: Comparison of the selected schema inference approaches

	Sevilla Ruiz et al. [92]	**Klettke et al.** [94]	**Baazizi et al.** [65]	**Izquierdo and Cabot** [98]	**Frozza et al.** [89]	**MM-infer** [4]
Inference Process	MapReduce + MDE	Fold into graph	Reduction in Apache Spark	MDE	Aggregation + fold into graph	Aggregation in Apache Spark
Scalable design	Yes	Yes	Yes	Yes	No	Yes
Scalable implementation	Yes	No	Yes	No	No	Yes
Implementation	Eclipse bundle	Spring Boot application	Apache Spark application in Scala	Eclipse bundle	Node.js web application	Apache Spark application in Java
Input format	Aggregate-oriented NoSQL data	JSON	JSON	JSON web service responses	Extended JSON	Multi-model data
Input type	Multiple kinds	Single kind	Single kind	Multiple kinds	Single kind	Multiple databases
Output format	NoSQL Schema model	JSON Schema	Custom textual type language	Ecore model	JSON Schema	ER, UML, JSON Schema, XML Schema, Categorical schema
Schema root	Entity	Record	Record	Entity	Record	Property
Extended JSON	No	No	No	No	Yes	Yes
Tuple	No	No	No	No	No	Yes
Set	No	No	No	No	No	Yes
Map	No	No	No	No	No	Yes
Optional	Yes	Yes	Yes	No	Yes	Yes
Union type	No	Yes	Yes	No	Yes	Yes
Order preserving	No	No	No	No	No	Yes
Identifiers	No	No	No	No	No	Yes
References	Partial	No	No	No	No	Yes
Complex IC	No	No	No	No	No	Partial
Data redundancy	No	No	No	No	Partial	Yes

also arbitrarily overlap (intersect), i.e., some data are partially or completely redundant. Moreover, we combine models that often have contradictory features and are based on different principles, e.g., aggregate-ignorant/oriented, order-ignorant/preserving, or allow different types of identifiers and references. Finally, we have to consider not only the features of the particular models, but other features resulting from their combination, e.g., cross-model references and cross-model data redundancy. All this brings a new dimension of complexity to the problem.

I4: *Unification and abstraction of complex data types.* At first sight, popular data models seem to approach common data structures (i.e., tuple, list, set, and map) in a uniform way. A closer look shows that this is not the case. For example, the column model explicitly distinguishes between all these structures, whereas the JSON document model explicitly considers only a list and a nested object. The tuple and set are implicitly represented by a list and the map as a nested object naturally containing mainly optional properties. In order to be able to infer a multi-model schema, we need not only the unification of the corresponding constructs across data models, but also their appropriate and not too general abstraction.

I5: *Scalability.* The proposed approach should be able to handle large volumes of multi-model data, but the inability to scale horizontally is a limitation of many observed approaches. Moreover, the approaches work with a large logical unit of data at a time. In particular, in one step the algorithms merge two records. If the data contains a large number of optional properties, the merged record is always more complex and an increasingly complex schema is continuously propagated to the next stages. Conversely, if we are working with a suitably small logical unit, e.g., merging schemas at the level of individual properties, then the large number of optional properties in the data has no effect on the structure of the merged single-property schema. Hence, algorithms working with a smaller logical unit of data can be significantly more efficient and scalable.

I6: *Ordering of properties.* The order of properties is a natural feature of several popular data models. For example, we can consider the order of elements in an XML document or the order of elements within a JSON array. Since multi-model data combines the features of each data model, the schema inference algorithm should be able to infer the order of features and elements.

I7: *Data redundancy*, i.e., the ability to represent the same data by various logical models, is another feature typical for multi-model data. A potential redundancy inference could improve query performance, e.g., to enable alternative query evaluation strategies.

From a more general point of view, the problem of inference of a multi-model scheme also involves the following open questions:

I8: *Fetching data.* Currently, schema inference approaches are tightly bound to a particular database system that implements a particular variant of the data model. As a result, the approaches may not be directly applicable

for schema inference in another system that supports the same data model. In addition, existing approaches often access data in a specific way. An optimal algorithm should be independent of the chosen database system and should also allow for different ways of retrieving data.

I9: *Universal processing.* The foundation of an ideal algorithm should be system-independent, e.g., exploiting unification of data model constructs. However, this universal algorithm can be based on so-called *wrappers* that implement the features of individual database systems and convert the input of the algorithm, e.g., structural information of data, into a unified form.

I10: *Schema representation.* Various graphical or textual languages are used to represent the (inferred) schema – for example ER [31], UML [32], DTD [71], XML Schema [104, 105], JSON Schema [106] and others. However, these formats are insufficient for representing the schema of multi-model data because they do not reflect all of its structural features.

0.4.4 Contribution: Framework MM-infer

So far, we have discussed the related work, the applicability of verified schema inference approaches to multi-model data, and the resulting open questions. We now turn to a commentary on our approach and explain the connection to the rest of the project.

At first sight, the JSON format appeared to be comprehensive enough to cover the structural features of multi-model data. At the same time, schema inference approaches over the collections of JSON documents scale very well and are capable of handling Big Data, and therefore seem like good candidates for extensions towards multi-model schema inference. However, during our research of existing ★ approaches [16], we found out that a selected single-model approach will not only need to be extended but also combined with features of other approaches.

In the thesis of Ivan Veinhardt Latták [107] (supervised by Pavel Koupil), we tried to extend a chosen approach to support multi-model data. In particular, the work resulted in the design of an algorithm based on the approaches of Baazizi et al. [65] and Sevilla Ruiz et al. [92]. However, not even this algorithm is sufficient for schema inference for multi-model data. Hence, based on the identification of the drawbacks of the proposed approach, an extended list of requirements that an ★ optimal schema inference algorithm should satisfy was created [91]. In addition, the paper also experimentally compares the dynamic features of the approaches from Table 0.4.

The core drawback of the proposed algorithms is the lack of coverage of all structural features of different popular data models. Therefore, we performed a thorough analysis of selected popular data models and introduced a unification of semantically similar features (challenge I4). We found out that the basis of the unified model is the name/value pair. We refer to this pair as a property. Based on the type of the value part, we divide properties into *simple* and *complex*. A simple property has a scalar value, while the value of a complex property can be a (homogeneous or heterogeneous) array, a set, or a map. Note that we only need to consider these three examples if we consider tuples as a special case of arrays

and nested structures as a special case of maps (see Paper IV). The unification of ★ the constructs allows us to better grasp the general features of multi-model data, and we are able to introduce universal data structures to describe the schema and data, as well as a general horizontally scalable algorithm that considers the varied features of data models from the very beginning and infer the true multi-model schema (challenge I3).

The foundation of our approach consists of two data structures – the *Record Schema Description* (RSD) and the *Property Domain Footprint* (PDF) (see Paper ★ IV). The RSD describes the structure of a single property (or a record as a special case of a root property) including the name of the property, the frequency of occurrence, the relationship to other properties within the same record (i.e., the parent/child hierarchy), the order of the children properties (challenge I6), and other features. The second data structure, PDF, allows us to describe all values of a given property in a compact way, including, e.g., uniqueness, multiplicity, occurrence of a particular value etc. This allows us to efficiently compare the active domains of two properties (challenge I2).

Altogether, we proposed three universal algorithms for inferring a structure or integrity constraints in multi-model data (challenge I9):

- The *Record-Based Algorithm* (RBA) infers a schema by gradually merging RSDs that describe the schema of a record.

- The *Property-Based Algorithm* (PBA) was designed to test the hypothesis that processing smaller logical units of data allows for a more scalable approach. We validated our hypothesis using experiments (see Paper IV) and ★ we confirmed that PBA is much more suitable for dealing with larger volumes of highly structured data (challenge I5), while RBA is more suitable for small volumes of flat data (i.e., aggregate-ignorant systems).

- The *Candidate Miner* extracts candidates for basic integrity constraints, e.g., identifiers, references, ranges of values (challenge I1), and in addition allows us to detect redundancy in the data (challenge I7). Note that we only retrieve candidates, because we describe the active domain using, among other things, Bloom filters, which with certain probability return a false positive result [108]. The user can then validate the selected candidates or the candidates are verified by an algorithm.

All listed algorithms are implemented using the Apache Spark framework and are therefore horizontally scalable.

Finally, the algorithms were verified using the prototype implementation *MM-infer* [109],[28] which currently supports schema inference for data stored in *PostgreSQL* (a representative of a multi-model schema-mixed database), *MongoDB* (a representative of a document schema-free database), and *Neo4j* (a representative of a graph schema-free database). Access to each database and retrieval of the data from which the schema is subsequently inferred is done via system-specific wrappers (challenge I8). And the tool also allows the selection of the format for representing the resulting inferred schema (challenge I10).

[28] https://www.ksi.mff.cuni.cz/~koupil/mm-infer/index.html

0.5 Evolution Management

Despite the correctness of the database schema design, sooner or later user requirements may change. A process that reflects the change of user requirements into the schema, data and related queries, integrity constraints, and data storage strategy while maintaining the integrity of the overall system is referred to as *evolution management*. Currently, this represents one of the most complex challenges [110].

Evolution management involves three main tasks: (1) managing structural changes of data, either explicitly applying *schema modification operations* (SMOs) or implicitly by reverse engineering, (2) propagating structural changes to data, i.e., data migration strategies, and (3) propagating such changes to queries. Moreover, there are, e.g., benchmarks that evaluate the effectiveness of data migration [111] and querying operations [112, 113].

Currently, there is a number of evolution management approaches for relational DBMS [20, 114, 115] or NoSQL systems [110, 116, 117, 118, 119] and the first approaches proposed for multi-model DBMS [120, 121] are emerging. However, the existing solutions have various limitations, targeting only a small fraction of data models [122], and partially or completely ignoring integrity constraints and features arising from the combination of multiple data models [117].

In this section, we elaborate on five selected promising approaches to evolution management. Based on the analysis of the selected approaches, we discuss a set of open questions and challenges in evolution management of multi-model data. Finally, we present our evolution management approach.

0.5.1 Closely Related Approaches

We first review two approaches [19, 20] that build an evolution management approach on an abstract data model. Next, we discuss approaches applicable to NoSQL database systems [110, 120, 116] that, in addition to schema changes and their propagation to the data, also consider other aspects of evolution management. Finally, we mutually compare the selected approaches.

Evolution Management in CGODD

The idea of utilising category theory in evolution management is not new. The beginnings can be found, e.g., in the academic approach [19] (see Subsection 0.3.2). Let us recall that (1) the approach is based on the notion of a *typegraph*, representing a schema and corresponding to an object T of a functor category $\mathbf{G}$ (see Lemma 1), (2) the data is represented by a graph corresponding to an object G of a functor category $\mathbf{G}$, and (3) a data instance is a morphism $Inst : G \to T$ associating data with the schema.

In this approach, schema changes are implemented using a basic set of (SMOs) based on graph pattern matching as follows:

- The *single addition* operation, denoted `ADD_S`, retrieves a single part of the data G corresponding to the pattern P_T and extends it according to the pattern Q_T. Categorically speaking, this corresponds to a pushout (see Definition 17) from $Q_T \xleftarrow{f} P_T \xrightarrow{m} G$, where $m : P_T \to G$ is a graph pattern

matching and morphism $f : P_T \to Q_T$ represents the intended modification. Note that the pattern Q_T may describe a more complex extension, not just an addition of a single property.

- The *full addition* operation, denoted `ADD_F`, finds all occurrences of the pattern P_T in the data G and extends them according to the pattern Q_T. Categorically speaking, this is the construction of a limit [19, 123].
- The *single deletion* operation, denoted `DEL_S`, finds a single part of the data G corresponding to the pattern Q_T and preserves only the subpart corresponding to the pattern P_T.
- The *full deletion* operation, denoted `DEL_F`, finds all occurrences of the pattern Q_T in the data G, and preserves only the data corresponding to the pattern P_T. Categorically speaking, this is again a construction of a limit.

Example 0.14. Figure 0.12 illustrates an example of an extension of a particular address in data G by the vertex named "Czechia", i.e., a single addition operation. The intended modification of the data is represented by the morphism $f : P_T \to Q_T$, i.e., we join edgewise the vertex named "Czechia" to the pattern P_T (indicated in green). Graph pattern matching is represented by the morphism $m : P_T \to G$ (indicated in blue). Note that in this particular case there are two morphisms $m, m' : P_T \to G$, each mapping the graph pattern P_T to a different part of the data G (note that only the morphism m is illustrated). Finally, the data extension is categorically implemented as a pushout from $Q_T \xleftarrow{f} P_T \xrightarrow{m} G$ (indicated in red), i.e., we extend the data from G corresponding to the pattern P_T by Q_T, thus obtaining the modified data G'. □

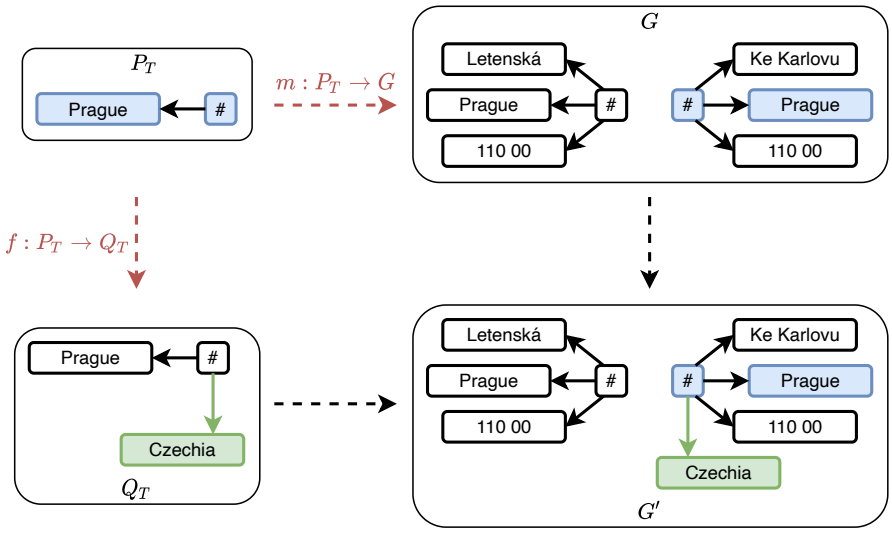


Figure 0.12: An example of a single addition (i.e., a pushout)

The schema changes are immediately propagated to the data (i.e., a strategy commonly known as an *eager data migration*). Note that the SMOs can be combined and concatenated, resulting in complex operations expressing, e.g., union and intersection. Finally, the concatenation of operations can also represent projection, join, and difference, hence the same idea is also applicable to querying based on graph pattern matching.

Evolution Management of Relational Data (Spivak et al.)

Yet another academic approach [20], this time relying on the logical model introduced in Subsection 0.3.2, allows relational data migration according to schema changes. Moreover, the approach enables a change of data representation at the logical layer.

Let us recall that the schema is represented as a free category $\mathbf{C}$ (see Definition 5) and the corresponding data instance is given by the set-valued functor $Inst_{\mathbf{C}} : \mathbf{C} \to \mathbf{Set}$. In addition, there is a category of all instances corresponding to the schema $\mathbf{C}$, i.e., the functor category $\mathbf{Set}^{\mathbf{C}}$ (see Lemma 1 and Example A.8).

In this approach, an SMO translating the schema $\mathbf{C}$ into $\mathbf{D}$ categorically corresponds to the functor $F : \mathbf{C} \to \mathbf{D}$. These functors can be used to represent operations, e.g., rename, delete, copy, intersection, and union for both kinds and properties. Note that the approach does not implement the operation addition, but it only introduces SMOs over existing data.

The schema changes, represented by the functor $F : \mathbf{C} \to \mathbf{D}$, are then propagated to the data utilising the so-called *data migration functors*:

- The functor $\Delta_F : \mathbf{Set}^{\mathbf{D}} \to \mathbf{Set}^{\mathbf{C}}$ propagates operations rename, delete, and copy to the data at the level of a kind or a property. Note that although the functor $F : \mathbf{C} \to \mathbf{D}$ transforms $\mathbf{C}$ into $\mathbf{D}$, the functor Δ_F propagates changes in the opposite direction (i.e., pointing "backwards"). Categorically, this is the principle described as "pulling back along functor F" and functor Δ_F is referred to as a *pullback* [20].[29]

- The functor $\Pi_F : \mathbf{Set}^{\mathbf{C}} \to \mathbf{Set}^{\mathbf{D}}$ propagates operations of intersection and (again) renaming of both a kind or a property. Categorically speaking, the functor Π_F corresponds to the right adjunct [20, 123] of the pullback functor Δ_F and is referred to as the *right pushforward*.

- The functor $\Sigma_F : \mathbf{Set}^{\mathbf{C}} \to \mathbf{Set}^{\mathbf{D}}$ propagates operations of union and (once again) renaming of both a kind or a property. Categorically, the functor Σ_F corresponds to the left adjunct [123] of the pullback functor Δ_F and is referred to as the *left pushforward*.

Example 0.15. Figure 0.13 illustrates an example of a schema change represented by the functor $F : \mathbf{C} \to \mathbf{D}$ (see Figure 0.13 (a)), i.e. merging kinds Audiobook and Book into a single kind Product.

Pullback functor Δ_F (see Figure 0.13 (b)) performs copy, delete and rename operations, i.e., the data from kind Product is first copied into two new kinds, the kinds are renamed to Audiobook and Book, and finally a property Pages is removed from Audiobook and a property Length is removed from Book.

Right pushforward Π_F (see Figure 0.13 (c)) implements the intersection and rename operations. First, a new kind is created by the intersection of Audiobook and Book,[30] and subsequently the newly created kind is renamed to Product.

The left pushforward Σ_F (see Figure 0.13 (d)) implements the union and rename operations. First, a new kind is created by merging Audiobook and

[29] Note that the pullback is an overloaded notion in category theory. It denotes not only the pullback introduced in the Definition 16, but also other constructs [123].

[30] Note that the identifiers of the corresponding records are merged.

Book, and then the new kind is renamed to Product. Note that left pushforward performs a so-called *skolemization* [124], which can be roughly understood as utilisation of null meta-values. □

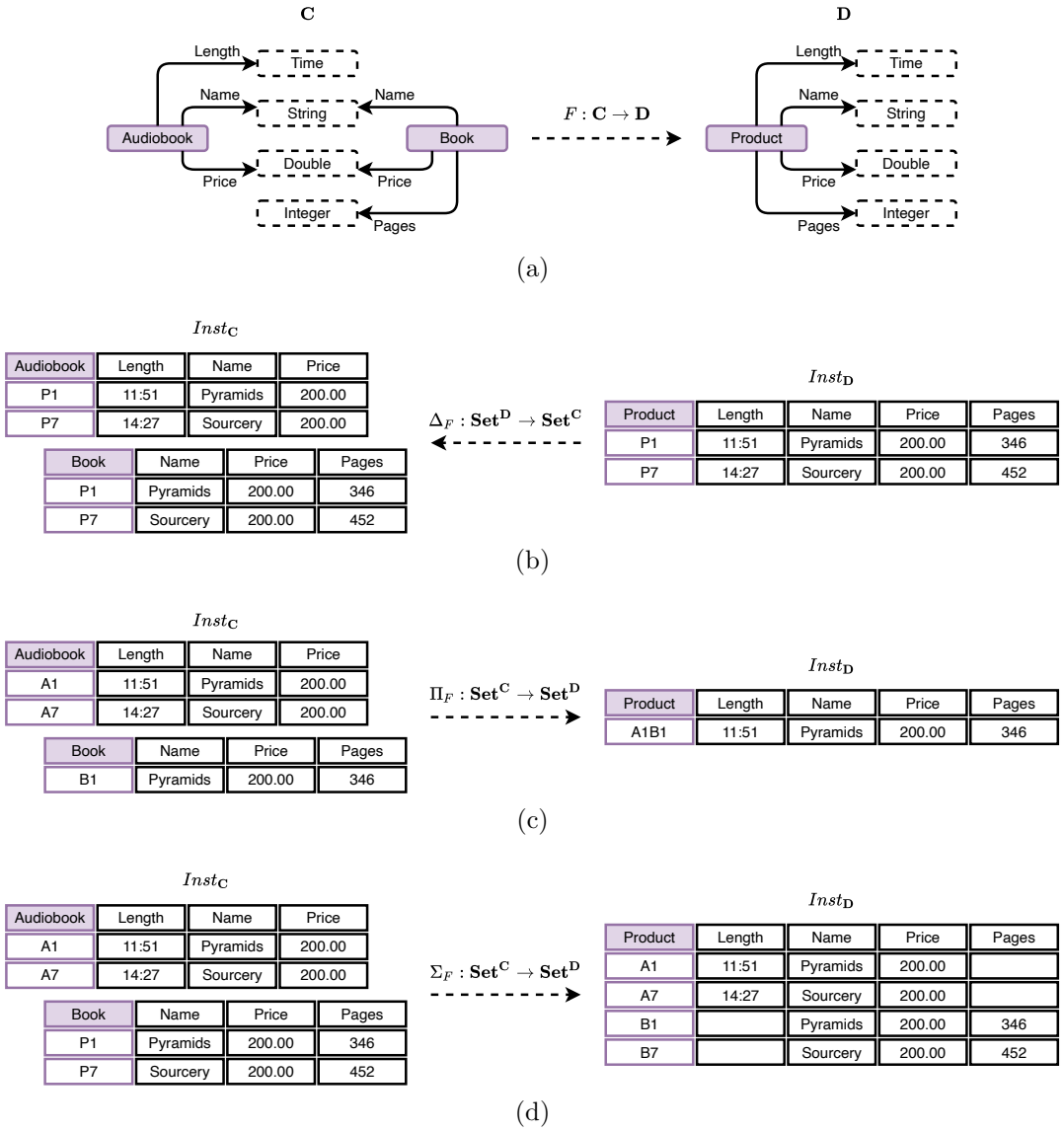


Figure 0.13: An example of schema mapping functor (a) inducing pullback (b), right pushforward (c) and left pushforward (d), i.e., data migration functors

The approach also allows to change the logical representation of data, in particular from relational model to RDF data and vice versa. Categorically, the data transformation is based on the application of the category of elements (also known as Grothendieck construction) [20, 123].

Recently, the authors of the paper [57] proposed an extension of the approach [20] to support multi-model data. In particular, the extension allows the representation and migration of data logically represented as JSON documents or graphs. However, as far as we know, this is only a theoretical extension and it is not implemented in CQL.

Darwin

Among the many academic prototypes, Darwin [110, 125] represents a family of approaches targeting schema management and data migrations. Basic features include semi-automatic declaration of schema changes [126], propagation of changes to data [127], inference of versioned schema including the historical sequence of versions [95], and various optimisation proposals [122, 128, 129, 111].

The authors propose a general programming language, the *Schema Evolution Language* (SEL), which allows declaring the following platform-independent SMOs:

- The *add* operation appends a property to every (selected) record of the particular kind or adds a new kind.
- The *delete* operation removes the property from each (selected) record of the particular kind or performs the removal of the kind.
- The *rename* operation changes the name of the property in all (selected) records of the particular kind or renames the kind.
- The *move* operation moves the property from each (selected) record of the input kind to the particular records of the destination kind.
- Similarly, the *copy* operation makes a copy of each (selected) record in a particular kind and inserts the copy into the particular record of the destination kind.

In addition, the operations can be assigned specific values (e.g., an added value in the case of the operation add) or filtering conditions specifying the selection of a subset of records of a particular kind. Also note that SEL expressions are subsequently translated into a domain specific language.

As for the propagation of SMOs to the data, the approach allows for multiple data migration strategies:

- *Eager strategy*: The changes are propagated immediately after the SMO is executed.
- *Lazy strategy*: The migration to a new version is delayed until a new version is required to perform a data management task.
- *Incremental strategy*: A hybrid strategy that completely migrates data, e.g., after a predetermined number of SMOs have been performed. Otherwise, the data is migrated lazily.
- *Predictive strategy*: A hybrid strategy that tracks the number of accesses to data. Frequently accessed data is migrated immediately after SMO execution, while the remaining data is migrated lazily.

The complete solution also includes a set of so-called *composition rules* [130], which allow to compose SMOs. Since typically composed SMOs are executed repeatedly, caching is utilised, which significantly improves the performance of data migration of different versions [131]. In addition, the authors propose a utilisation of composition rules for query rewriting [132].

Finally, the family of approaches includes MigCast [122] and EvoBench [111]. The former tool allows comparing different data migration strategies, e.g. in terms of operational costs, and the latter is a benchmark for schema evolution. The complete set of tools is then available as a docker container,[31] currently supporting a number of NoSQL database systems such as, e.g., MongoDB, Couchbase, Cassandra and ArangoDB.

MM-evolver

The first approach targeting evolution management of multi-model data is the academic prototype MM-evolver [120]. This tool allows propagating of SMOs to data represented by different and interconnected models and additionally considers (in a limited way) cross-model references and redundancy.

The approach implements the set of platform-independent SMOs supported by Darwin, and in addition introduces a special operation to add/delete a reference. Note that internally, the operation differs from the Darwin system in propagation to multiple data models as well as propagation to references. For example, if a property is removed, all references (including intra- and inter-model) are removed as well.

Similar to the Darwin approach, the authors propose a general programming language for declaring platform-independent SMOs, the *Multi-Model Schema Evolution Language* (MMSEL). As in the previous case, MMSEL expressions are subsequently translated into a domain specific languages.

Finally, schema changes are immediately propagated to the data, i.e., only the eager strategy is applied.

Orion

Recently, the family of academic approaches for representation [30] and management [92, 93] of data in NoSQL systems has been extended with the language Orion [116], which allows declaring SMOs over the U-Schema logical model (see also Subsection 0.3.2).

From a logical unit perspective, schema modification operations (SMOs) declared by Orion can be classified as kind-level, version-level, and property-level. Kind-level operations include common operations such as *add*, *delete*, *rename*, *extract* (i.e., copy), and in addition:

- The *split* operation splits the set of properties of one kind to create two new kinds, the original kind being removed.
- The *merge* operation merges the properties of two kinds to create a new kind, replacing the original kinds.

Version-level operations, performed at the level of record version, include:

- The *delvar* operation removes a specific record version from the schema, and the change is propagated to the data by deleting all records corresponding to the deleted version.

[31] https://sites.google.com/view/evolving-nosql/tools/darwin

- The *adapt* operation also removes the record version from the schema, but the corresponding records are converted (adapted) to another, selected record version.
- The *union* operation merges two versions of the records into one of them.

Finally, property-level operations are implemented over simple and complex properties (including references as a special kind of a property), and allow for the common operations such as *add*, *delete*, *rename*, *extract*, and *move*. They also include:

- The *nest* operation allows nesting properties, i.e., moving the selected property to the nested complex property.
- The *unnest* operation, which is inverse to *nest*.
- The *cast* operation allowing to change the data type of a simple property or a reference.
- The *promote* operation implementing the addition of an attribute to the key.
- The *demote* operation, on the other hand, allows the attribute to be removed from the key.
- The *mult* operation implementing a cardinality change on a reference or a complex property (e.g., an array).
- The *morph* operation allows to replace a reference with a nested complex property (aggregate) and vice versa.

As in the previous cases, the Orion language allows the usage of filter conditions to specify the subset of records over which the SMO is performed. Once again, SMOs are platform-independent and translated into domain-specific languages, and the schema changes are propagated eagerly.

Currently, the supported systems include MongoDB, Cassandra, and Neo4j, i.e., representatives of document, column, and graph databases. However, the the U-Schema model allows the extension towards the support of key/value and (a partial support of) relational DBMSs.

Although the authors attempt to create a database-independent language for describing schema changes, the language is burdened by the integration, (but) not the unification of data models in U-Schema. As a consequence, the language is complex as it has to take into account all model-specific constructs and properties of underlying data models. Hence, the extensibility of the approach is limited.

Comparative Summary

Table 0.5 illustrates a comparison of the described evolution management approaches. It also includes a comparison with our proposed approach (MM-★ evocat), but it is addressed in a separate chapter (see Subsection 0.5.3 and Paper V). All the compared approaches implement a set of SMOs, yet they differ in the choice of the particular supported operations (see Table 0.6).

Table 0.5: A comparison of features in the selected evolution management approaches

	CGOOD [19]	**Spivak et al.** [20]	**Darwin** [110]	**MM-evolver** [120]	**Orion** [116]	**MM-evocat** [5]
Schema modification	Yes	Yes	Yes	Yes	Yes	Yes
Data migration	Eager	Eager	Eager, lazy, hybrid	Eager	Eager	Eager
Version jumps	No	No	Yes	No	No	No
Propagation to IC	No	No	No	Partial	Partial	Yes
Schema inference	No	No	Tracked, untracked	No	Untracked	No
Query rewriting	No	No	Yes	No	No	No
Benchmarking	No	No	Yes	No	No	No
Self-adaptation	No	No	No	No	No	No
Data models	Relational, object, object-relational	Relational, RDF	Graph, key/value, document, columnar	Relational, graph, key/value, document, columnar	Relational, graph, key/value, document, columnar	Relational, array, graph, RDF, key/value, document, columnar
Multi-model	No	No	No	Yes	No	Yes

Table 0.6: A comparison and classification of supported SMOs in the selected evolution management approaches

	CGOOD [19]	**Spivak et al.** [20]	**Darwin** [110]	**MM-evolver** [120]	**Orion** [116]	**MM-evocat** [5]
Model-level	-	-	-	-	-	add, delete, move, copy
Kind-level	add, delete	delete, rename, copy, intersection, union	add, delete, rename	add, delete, rename	add, delete, extract (copy), rename, split, merge	add, delete, rename, copy, move, group, ungroup
Property-level	add, delete	delete, rename, copy, intersection, union	add, delete, move, copy, rename	add, delete, move, copy, rename	add, delete, move, extract, rename, cast, nest, unnest	add, delete, rename, copy, move, group, ungroup, union, split
Identifier-level	-	-	-	-	promote, demote	addId, dropId
Reference-level	-	-	-	reference	morph	addRef, dropRef
Cardinality-level	-	-	-	-	mult	changeCardinality
Version-level	-	-	-	-	delvar, adapt, union	-

As we can see, the vast majority of approaches only allow for propagating of changes to the data immediately. The exception is Darwin, which allows for other data migration strategies, namely lazy and hybrid. The support for different strategies is reflected in the optimisation capabilities, where once again only Darwin considers optimising data migration by composing SMOs, caching the composed SMOs, and performing lazy data migration efficiently.

The propagation of SMOs to integrity constraints is addressed marginally, with only MM-evolver and Orion addressing references or identifiers in a limited way. Similarly, propagation of SMOs to queries and benchmarking of SMOs are less common features.

Next, only Darwin and Orion allow for inference of versioned schema as an alternative to otherwise explicit schema evolution specified using SMOs. Moreover, in the case of Darwin, the historical sequence of schema versions can be inferred.

Finally, the selected approaches also differ in the extent and support of data models. While the categorical approaches are mainly bounded with aggregate-ignorant models (especially the relational model), aggregate-oriented models are commonly supported in the remaining approaches. However, only the MM-evolver approach considers a set of linked or overlapping data models, i.e., multi-model data. In the other cases, a disjunctive set of models is considered.

0.5.2 Open Questions and Challenges in Evolution Management

We believe that in order to be able to process evolution management for multi-model data, we need to address the limitations of existing approaches and extend them appropriately towards the full support of multi-model data. Therefore, we provide the following list of **challenges E1 – E11** in the area of evolution management of multi-model data.

E1: *Multi-model data.* The aspect of multi-model data brings new dimension of complexity to evolution management approaches. In addition to the model-specific features of (otherwise disjunctive) logical models, we must consider features arising from the combination of the models, such as cross-model references, cross-model embedding, cross-model integrity constraints, redundancy, and inconsistency in data. Moreover, we have to deal with often contradictory features of the models, e.g., aggregate-oriented/-ignorant, schema-full/-less/-mixed, and order-preserving/-ignorant.

E2: *Unification (abstraction) of data models.* Currently, there is a number of approaches that struggle with insufficient unification of underlying logical models. Hence the proposed schema modification language is complicated. It is important to unify, for example, the different approaches to the otherwise corresponding data structures (e.g., JSON object and a map), and to work uniformly with different forms of identifiers (i.e., simple, complex, multiple, and overlapping) and links between objects (e.g., embedding or references).

E3: *Propagation of SMOs to queries.* The propagation of schema changes to the data is only one of many efforts to adapt to changing user requirements.

Naturally, as the schema (i.e., a data structure) changes, the queries must be updated to remain syntactically and semantically correct [132].

E4: *Integrity constraints evolution management.* Currently, there are several approaches to modify the data structure in particular. A schema, however, also includes integrity constraints describing identifiers, references, and, e.g., complex business rules. As far as we know, only several approaches targeting relational DBMSs introduce so-called *integrity constraints modification operations* (ICMO) [114], which allow modification of complex business rules. For non-relational DBMSs, only changes to identifiers or references are supported to a limited extent [120, 116]. In addition, a new dimension of difficulty not observed in relational systems is the implicit management of integrity constraint changes in schema-free data, where integrity constraints must first be inferred and the historical sequence of changes in these constraints must be tracked.

E5: *Propagation of changes to the storage strategy.* Not only a new data structure, but also new queries may reflect the change in user requirements. In multi-model systems, we can respond to this change by adapting the logical representation of the data to maintain/improve the efficiency of query evaluation. However, a minimal number of approaches currently exist that deal with the change in logical data representation [130, 120].

E6: *Extraction of changes.* There is a number of possible methods for retrieving information about changes in user requirements. Most often, these are user-specified changes in the form of SMOs. This approach is particularly useful when dealing with schema-full data. In NoSQL systems, approaches that infer implicit versioned and chronologically ordered schema of data exists. Finally, changes in user requirements and data can be inferred by observing changes in queries. However, none of the solutions is trivial, especially in the case of multi-model data.

E7: *Integration of new data models and data formats.* A significant part of existing evolution management approaches assumes that the logical representation of data is time-invariant. Hence, in practice, we encounter approaches that support only a single data model or a limited set of disjunctive models. If additional data formats are needed, then (1) we need to represent this format by means of supported data formats and models, even if at the cost of relaxing the requirements for efficient data processing, or (2) allow the integration of new data models and formats.

E8: *Benchmarking.* In order to evaluate the effectiveness of the data migration process and to query the successfully migrated data, we need to be able to put a price on each operation and compare them with respect to each other. Currently, there are first drafts of a benchmark for multi-model querying [112, 113] and tools to determine the cost of data migration in polystores and multi-model DBMSs [111].

E9: *Intuitive naming of SMOs.* The contradictory features of underlying logical models in multi-model DBMSs can result in seemingly unexpected propagation patterns into (redundant) data. For example, the ungroup operation

changes the logical structure of data represented by aggregate-oriented approaches compared to the representation of aggregate-ignorant approaches in exactly the opposite way, i.e., in the former case there may be an inlining into the parent property within one kind and in the latter case there will ★ be merging of two kinds (see Paper V). Although the naming of the ungroup operation accurately reflects what happens in the aggregate-oriented approach, in the case of the aggregate-ignorant approach the naming is completely non-intuitive.

E10: *Data migration overhead.* Currently, there are several strategies to propagate schema changes to the data. Eager migration propagates schema changes to data immediately, regardless of the current DBMS workload. The lazy strategy propagates changes only when the current schema version is needed, introducing an overhead during (critical) data management tasks, similarly to the hybrid strategies, which combine features of eager and lazy strategies. However, the burden of data migration is still on the datastore regardless of the chosen strategy. The question is whether we must necessarily propagate all changes to the data at the physical level, i.e., stress the database system, or whether changes can only be propagated "virtually".

E11: *Involvement of artificial intelligence.* When a change in user requirements occurs, the user initiates the change in the data structure, e.g., by executing an SMO or by inferring the schema from the new data. However, the choice of the SMO may not always be optimal and reflect all user requirements. We believe that by engaging artificial intelligence to be trained based on user input (e.g., data writes, queries, typical system usage time etc.), we can (1) optimise the choice of the logical representation and the data schema with respect to the data management tasks to be performed, and (2) appropriately plan and select a data migration strategy, e.g., based on the DBMS workload at a specific time.

0.5.3 Contribution: Framework MM-evocat

So far, we have discussed the related work, the applicability of established approaches to evolution management towards multi-model data, and the resulting open questions. We now move on to comment on our approach and explain the connections with the rest of the thesis.

First, we analysed selected existing evolution management solutions introducing a platform-independent layer for dealing with multiple data models [19, 20, 110, 120, 116] and verified their applicability to multi-model data, identifying drawbacks of the selected solutions and outlining open questions.

Being inspired by existing solutions, our schema evolution approach is based on an appropriate abstraction of data models[32] (challenge E1) allowing extension towards the support of additional models (challenge E7), and we define a basic set of SMOs on top of this model. In addition, we addressed the question of whether it is more user-friendly (1) to take a minimalist approach, i.e., to define only basic

★ [32]Let us recall that this refers to the categorical model (first introduced in Paper II and
★ commented in Subsection 0.3.4) and its mapping to the underlying logical layer (see Paper III).

operations from which the user composes complex operations (e.g., see [19]), or (2) to add complex but frequently called operations to the basic set of SMOs (e.g., see [116]). We have opted for a combination of approaches, whereas SMOs[33] can be classified into three tiers:

1. *Basic operations.* Arbitrary modification of the unifying conceptual model can be achieved by any of the six basic operations, i.e., addObject, deleteObject, addMorphism, deleteMorphism, addId, dropId, or their combination. Nevertheless, although these are expressive operations, they are not very user friendly.

2. *Complex operations.* To enhance user experience, we introduce a set of complex operations, i.e., addRelationship, addList, addSet, addMap, addHierarchy, addStructure, addProperty, addRef, renameProperty, deleteRelationship, deleteList, deleteSet, deleteMap, deleteHierarchy, deleteStructure, deleteProperty, and dropRef. These operations internally combine basic operations (and implement this composition efficiently) and allow for modifications to the conceptual model in a user-friendly way.

3. *Frequently called operations.* Based on typical user requirements, we introduce a group of frequently called operations, i.e., copy, move, group, ungroup, union, split, and changeCardinality. These operations are internally composed of basic and/or complex operations.

We provide the most user-friendly SMOs in the Table 0.6. Note that our approach is the only one that allows SMOs in the (logical) model-level (challenge E5), i.e., we explicitly consider cross-model redundancy. Moreover, as a result of the unification of the underlying model constructs, the SMOs we propose are not platform dependent, i.e., there is no need to distinguish between, e.g., embedding or referencing – that is a logical layer detail (challenge E2).

The proposed SMOs are declared in the platform-independent Multi-Model Schema Evolution Language (MMSEL) (see Paper V). An obvious part of the ★ language is the ability to select a subset of records of a given kind over which to execute SMOs (i.e., a selection). Analogous to existing solutions, MMSEL expressions are translated into domain specific language (DSL) utilising so-called wrappers. Note that we also exploit a mapping between the conceptual and logical layers for translation into DSL. Compared to existing solutions for NoSQL and multi-model systems, our approach also propagates SMO to identifiers and references (challenge E4).

SMOs are propagated into the data using the following data transformation algorithms:

- The *Model-to-Category transformation* (see Paper III) is applicable for data ★ translation between logical and categorical layers. The algorithm first reads the input data from the logical model and inserts the records into a unifying data structure that structurally corresponds to the schema category (i.e., the unifying data structure).

[33]These operations are discussed in more detail in the journal article "A Unified Evolution Management of Multi-Model Data Using Category Theory", which is currently unfinished, hence unpublished. The expected completion is Q3 2022.

★ • The *Category-to-Model transformation* (see Paper III) converts the data from a categorical to a logical representation in three steps:

1. The *DDL algorithm* allows the translation of a unified schema into a platform-specific schema (i.e., it creates, e.g., statements CREATE KIND, ALTER KIND).
2. The *DML algorithm* creates a list of DML statements to store the data in the logical layer (e.g., INSERT INTO KIND).
3. Finally, the *IC algorithm* ensures specification of identifiers and references at the logical layer (e.g., ALTER KIND).

The main contribution of the algorithms is their universality for migration or changing the logical representation of data. Moreover, the input and output of data migration algorithms may be the data represented by a combination of logical models. Hence, we have multi-model to multi-model data migration (for ★ more details see Paper III). SMOs are propagated into the data by the eager strategy. Extending the support to other data migration strategies, e.g., lazy and hybrid strategies [110], forms our current and future work. In addition, we add the possibility of scheduling data migration with respect to the usual database system workload, i.e., data will be migrated not only when needed (i.e., lazy migration), but also proactively during lower workloads.

Furthermore, SMOs are classified into heavy and light operations. *Heavy operations* are based on the traditional concept, i.e., they are propagated to the data and to the mapping between the conceptual and logical layers, thereby increasing the workload on the side of the database system. The examples of heavy operations include delete, move, union, and split. On the contrary, *light operations* propagate changes only to the mapping, i.e., no immediate propagation to the logical representation of the data is required (challenge E10). Examples of light operations includes add, rename, and group. Note that some heavy ★ operations can be light under certain conditions and vice versa (see Paper V).

Categorically speaking, schema modification operations (SMOs) correspond to functors, i.e., structure preserving mappings between two (schema) categories (inspired by the approach [20]). Regarding data migration, we propagate SMOs that add new schema elements using pushouts (inspired by the approach [19]) and changes that only duplicate or add no new elements are propagated using a pullback functor and its right and left adjuncts (inspired by the approach [20]).

Finally, the proposed approach was verified in the academic prototype *MM-evocat* [5],[34] which currently supports evolution management and backwards propagation for data stored in *PostgreSQL* (a representative of a multi-model DBMS), *MongoDB* (a representative of a document DBMS), and *Neo4j* (a representative of a graph DBMS). The advanced evolution management tasks, e.g., propagating schema changes to queries (challenge E3), extracting changes from schema-less data and from queries (challenge E6), proposing an approach for cost estimation of data migration operations (challenge E8), and involving artificial intelligence in the schema and data evolution process (challenge E11), constitute our current and near-future work.

[34] https://www.ksi.mff.cuni.cz/~koupil/mm-evocat/index.html

Paper I

Categorical Management of Multi-Model Data

Irena Holubová[@1], Pavel Čontoš (Koupil)[1], Martin Svoboda[1]

Published in 25th International Database Engineering & Applications Symposium (IDEAS 2021) by Association for Computing Machinery
doi: 10.1145/3472163.3472166

[@] corresponding author, e-mail: irena.holubova@matfyz.cuni.cz

[1] Department of Software Engineering, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

Abstract

In this vision paper, we introduce an idea of a framework that would enable us to model, represent, and manage multi-model data in a unified and abstract way. Its core idea exploits constructs provided by category theory, which is sufficiently general but still simple enough to cover any of the logical data models used in contemporary databases. Focusing on promising features and taking into account mature and verified principles, we overview the key parts of the framework and outline open questions and research directions that need to be further investigated. The ultimate objective is to pursue the idea of a self-tuning system that would permit us to collapse the traditionally understood conceptual and logical layers into just a single model allowing for unified handling of schemas, data instances, as well as queries.

Keywords

• Multi-model data • Category theory • Data modeling

Paper II

Categorical Modeling of Multi-Model Data: One Model to Rule Them All

Martin Svoboda[@1], Pavel Čontoš (Koupil)[1], Irena Holubová[1]

Published in International Conference on Model and Data Engineering (MEDI 2021), Lecture Notes in Computer Science (vol. 12732) by Springer
doi: 10.1007/978-3-030-78428-7_15

[@] corresponding author, e-mail: martin.svoboda@matfyz.cuni.cz

[1] Department of Software Engineering, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

Abstract

Following the Gartner predictions, most of the DBMSs, both relational and NoSQL, have become multi-model. However, this functionality brought plenty of related issues. The core problem is how to design a multi-model application. The step from the conceptual layer to a set of distinct interlinked logical models is not straightforward.

In this paper, we propose an approach based on category theory, which provides a unified view of the data and a strong mathematical basis for their management. We propose a schema and instance categories covering popular models and we show how an ER model can be transformed to such a categorical layer. We also introduce the whole framework based on the categorical model and discuss open research issues.

Keywords

• Multi-model data • Category theory • Conceptual modelling

Paper III

A Unified Representation and Transformation of Multi-Model Data using Category Theory

Pavel Koupil[@1], Irena Holubová[1]

Published in Journal of Big Data (9, 61, 2022) by Springer Nature
doi: 10.1186/s40537-022-00613-3

[@] corresponding author, e-mail: pavel.koupil@matfyz.cuni.cz

[1] Department of Software Engineering, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

Abstract

The support for multi-model data has become a standard for most of the existing DBMSs. However, the step from a conceptual (e.g., ER or UML) schema to a logical multi-model schema of a particular DBMS is not straightforward.

In this paper, we extend our previous proposal of multi-model data representation using category theory for transformations between models. We introduce a mapping between multi-model data and the categorical representation and algorithms for mutual transformations between them. We also show how the algorithms can be implemented using the idea of wrappers with the interface published but specific internal details concealed. Finally, we discuss the applicability of the approach to various data management tasks, such as conceptual querying.

Keywords

• Multi-model data • Category theory • Model transformations

Paper IV

A Universal Approach for Multi-Model Schema Inference

Pavel Koupil[@1], Sebastián Hricko[1], Irena Holubová[1]

Accepted in Journal of Big Data by Springer Nature
doi: 10.1186/s40537-022-00645-9

[@] corresponding author, e-mail: pavel.koupil@matfyz.cuni.cz

[1] Department of Software Engineering, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

Abstract

The *variety* feature of Big Data, represented by *multi-model data*, has brought a new dimension of complexity to all aspects of data management. The need to process a set of distinct but interlinked data models is a challenging task.

In this paper, we focus on the problem of inference of a schema, i.e., the description of the structure of data. While several verified approaches exist in the single-model world, their application for multi-model data is not straightforward. We introduce an approach that ensures inference of a common schema of multi-model data capturing their specifics. It can infer local integrity constraints as well as intra- and inter-model references. Following the standard features of Big Data, it can cope with overlapping models, i.e., data redundancy, and it is designed to process efficiently significant amounts of data.

To the best of our knowledge, ours is the first approach addressing schema inference in the world of multi-model databases.

Keywords

• Multi-model data • Schema inference • Cross-model references • Data redundancy

Paper V

MM-evocat: A Tool for Modelling and Evolution Management of Multi-Model Data

Pavel Koupil[@1], Jáchym Bártík[1], Irena Holubová[1]

Manuscript under review

[@] corresponding author, e-mail: pavel.koupil@matfyz.cuni.cz

[1] Department of Software Engineering, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

Abstract

The arrival of so-called *multi-model data* has brought many challenging problems. The contradictory features of the combined models and lack of standardisation of their combination make the solution of the data management specifics highly complex.

In this paper, we focus on the problem of evolution management of multi-model data. With the changing user requirements, the schema and the data need to be adapted to preserve the expected functionality of a multi-model application. We introduce a tool *MM-evocat* based on utilising the category theory. We will show that the core of the tool, i.e., the categorical representation of multi-model data, enables us to grasp all the specifics of the individual models and their possible combinations. Its simple but powerful formal basis enables unique and robust support for evolution management.

Keywords

• Multi-Model Data • Evolution Management • Category Theory

Conclusion

This thesis introduces a general framework for modelling and management of multi-model data. Unlike the existing solutions, the framework is based on a mature formal background of a theory general enough to grasp the variety of all popular data models and to support different data management tasks, such as, e.g., data modelling, schema inference, data migration, and evolution management, in a unified way.

The main contributions are summarised as follows:

- *Unification of data models.* First, an extensive analysis of popular database systems and underlying data models was performed. Based on the results, we proposed a unification of related constructs occurring in various data models and brought them to the same (abstract) level. Hence, contrary to the existing solutions, we do not introduce any constructs tied only to a particular model.

- *Multi-model data modelling.* The proposed data modelling approach is general enough to allow a unified representation of popular data models and their combination at the conceptual level. To verify the completeness of the proposal, an algorithm for translating the ER schema into the proposed categorical representation was proposed. Finally, we provided a unified data representation that serves as a mediator for various data management tasks.

- *The bridge between the conceptual and logical layer.* We have proposed an approach that allows for mapping of the unified categorical schema to any (combination of) popular models supported in existing DBMSs, while the specific features of the logical layer are hidden from the user. Along with the mapping, we introduced data transformation algorithms that, among others, allow to realise data migration between different (combinations of) logical representations. The core idea was implemented in the academic prototype *MM-cat*.

- *Inference of the multi-model schema.* To the best of our knowledge, we have proposed the first approach dealing with the inference of a multi-model schema. In addition, the approach allows to infer a number of integrity constraints, e.g., (simple) identifiers, references (both intra- and inter-model), and to reveal partial and complete redundancy in the data (once again, both intra- and inter-model). Last but not least, exploiting the statistical analysis of the source data, the approach enables the detection and backward correction of errors in the data. The main idea of the proposal was implemented as the academic prototype *MM-infer* and experimentally verified.

- *Evolution management and correct propagation of changes.* Having the unified conceptual layer, schema modifications within and across multiple logical models are reduced to modifications of the unified representation and its mapping to the logical layer. For this purpose, we proposed a several sets of SMOs, together with the respective propagation of changes

via transformation algorithms between the logical and unified layers. The main idea was implemented in the academic prototype *MM-evocat*.

Finally, let us note that the unification of models allows the framework to be applicable to multi-model data represented both polystores and multi-model systems, as well as to single-model systems. Even in the latter case of single-model data the proposed approaches also bring innovations and extensions thanks to the unified and general view of the respective data management tasks.

Current and Future Research

★ Besides extensions to existing components of the framework (see Paper III and
★ Paper IV) we are currently working on additional features:

- *Conceptual query language.* The level of abstraction of the categorical approach allows us to define a conceptual query language. Utilising the ideas of decomposition and mapping, any conceptual expression can be decomposed and further translated into particular query expressions at the logical level. Moreover, since a category can be seen as a special type of a multigraph, the categorical query language can be inspired by graph pattern matching, as known from existing graph languages such as, e.g., Cypher, and SPARQL.

- *Conceptual query evaluation plan.* The knowledge of a unifying schema and its decomposition allows the construction of multiple query evaluation strategies. Similar to single-model systems, there is an opportunity for creating multiple query execution plans and selecting the optimal query evaluation strategy. Moreover, we can exploit the natural properties of multi-model data, e.g., cross-model redundancy in the data, which allows for higher variability in query evaluation strategies.

- *Self-Adapting Evolution Management.* The combination of the variety of data formats and the continuous changes in the data bring a huge challenge for administrators of (not only multi-model) database systems. Our future goal is first to extend the existing evolution management proposal with the propagation of changes to queries and then to focus on the area of autonomous management of rapidly changing multi-model Big Data, as envisioned in our paper [133].

Bibliography

[1] Irena Holubova, Pavel Contos, and Martin Svoboda. Categorical Management of Multi-Model Data. In *25th International Database Engineering & Applications Symposium*, Ideas 2021, page 134–140, New York, NY, USA, 2021. Association for Computing Machinery.

[2] Martin Svoboda, Pavel Čontoš, and Irena Holubová. Categorical Modeling of Multi-Model Data: One Model to Rule Them All. In *Model and Data Engineering*, Lecture Notes in Computer Science, pages 190–198, Cham, 2021. Springer International Publishing.

[3] Pavel Koupil and Irena Holubová. A unified representation and transformation of multi-model data using category theory. *J. Big Data*, 9(1):1–49, 2022.

[4] Pavel Koupil, Sebastián Hricko, and Irena Holubová. A Universal Approach for Multi-Model Schema Inference. *J. Big Data (accepted)*, 2022.

[5] Pavel Koupil, Jáchym Bártík, and Irena Holubová. MM-evocat: A Tool for Modelling and Evolution Management of Multi-Model Data.

[6] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *Commun. ACM*, 13(6):377–387, 1970.

[7] Pramod J Sadalage and Martin Fowler. *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. Pearson Education, 2013.

[8] Boyan Kolev, Raquel Pau, Oleksandra Levchenko, Patrick Valduriez, Ricardo Jiménez-Peris, and José Orlando Pereira. Benchmarking Polystores: The CloudMdSQL Experience. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 2574–2579, New York, NY, USA, 2016. IEEE.

[9] Jennie Duggan, Aaron J Elmore, Michael Stonebraker, Magda Balazinska, Bill Howe, Jeremy Kepner, Sam Madden, David Maier, Tim Mattson, and Stan Zdonik. The bigdawg polystore system. *ACM Sigmod Record*, 44(2):11–16, 2015.

[10] Rana Alotaibi, Bogdan Cautis, Alin Deutsch, Moustafa Latrache, Ioana Manolescu, and Yifei Yang. ESTOCADA: towards scalable polystore systems. *Proceedings of the VLDB Endowment*, 13(12):2949–2952, 2020.

[11] Jiaheng Lu, Irena Holubová, and Bogdan Cautis. Multi-model Databases and Tightly Integrated Polystores: Current Practices, Comparisons, and Open Challenges. In *Proc. of CIKM '18*, pages 2301–2302. Acm, 2018.

[12] Jiaheng Lu, Zhen Hua Liu, Pengfei Xu, and Chao Zhang. UDBMS: Road to Unification for Multi-model Data Management. In *ER '18 Workshops*, volume 11158 of *Lecture Notes in Computer Science*, pages 285–294, Cham, 2018. Springer.

[13] Jiaheng Lu and Irena Holubová. Multi-Model Databases: A New Journey to Handle the Variety of Data. *ACM Computing Surveys*, 52(3), 2019.

[14] Irena Holubová, Martin Svoboda, and Jiaheng Lu. Unified Management of Multi-model Data. In *Proc. of ER '19*, pages 439–447. Springer, 2019.

[15] Heikki Mannila and Kari-Jouko Räihä. *The design of relational databases*. Addison-Wesley Longman Publishing Co., Inc., 1992.

[16] Pavel Contos and Martin Svoboda. JSON Schema Inference Approaches. In Georg Grossmann and Sudha Ram, editors, *Advances in Conceptual Modeling - ER 2020 Workshops CMAI, CMLS, CMOMM4FAIR, CoMoNoS, EmpER, Vienna, Austria, November 3-6, 2020, Proceedings*, volume 12584 of *Lecture Notes in Computer Science*, pages 173–183. Springer, 2020.

[17] Saunders Mac Lane. *Categories for the working mathematician*, volume 5. Springer Science & Business Media, 2013.

[18] Lippe, E. and Ter Hofstede, A. H. M. A Category Theory Approach to Conceptual Data Modeling. *RAIRO Theor. Informatics Appl.*, 30(1):31–79, 1996.

[19] Chris Tuijn and Marc Gyssens. CGOOD, a Categorical Graph-oriented Object Data Model. *Theoretical Computer Science*, 160(1):217–239, 1996.

[20] David I Spivak and Ryan Wisnesky. Relational Foundations for Functorial Data Migration. In *Proceedings of the 15th Symposium on Database Programming Languages*, Dbpl 2015, pages 21–28, New York, NY, USA, 2015. Association for Computing Machinery.

[21] Joshua Shinavier and Ryan Wisnesky. Algebraic property graphs. *arXiv preprint arXiv:1909.04881*, 2019.

[22] Benjamin C Pierce. *Basic Category Theory for Computer Scientists*. MIT press, 1991.

[23] F William Lawvere and Stephen H Schanuel. *Conceptual Mathematics: a First Introduction to Categories*. Cambridge University Press, 2009.

[24] Michael J Healy, Richard D Olinger, Robert J Young, Shawn E Taylor, Thomas Caudell, and Kurt W Larson. Applying Category Theory to Improve the Performance of a Neural Architecture. *Neurocomputing*, 72(13-15):3158–3173, 2009.

[25] Dan Shiebler, Bruno Gavranović, and Paul Wilson. Category Theory in Machine Learning. *arXiv preprint arXiv:2106.07032*, 2021.

[26] Irena Holubova, Pavel Contos, and Martin Svoboda. Multi-Model Data Modeling and Representation: State of the Art and Research Challenges. In *25th International Database Engineering & Applications Symposium*, Ideas 2021, page 242–251, New York, NY, USA, 2021. Association for Computing Machinery.

[27] Paolo Atzeni, Francesca Bugiotti, Luca Cabibbo, and Riccardo Torlone. Data Modeling in the NoSQL World. *Computer Standards & Interfaces*, 67:103–149, 2020.

[28] Jeremy Kepner, Julian Chaidez, Vijay Gadepally, and Hayden Jansen. Associative Arrays: Unified Mathematics for Spreadsheets, Databases, Matrices, and Graphs. *CoRR*, abs/1501.05709, 2015.

[29] Eric Leclercq and Marinette Savonnet. TDM: A Tensor Data Model for Logical Data Independence in Polystore Systems. In *Heterogeneous Data Management, Polystores, and Analytics for Healthcare*, pages 39–56, Cham, 2019. Springer International Publishing.

[30] Carlos J. Fernández Candel, Diego Sevilla Ruiz, and Jesús J. García-Molina. A unified metamodel for NoSQL and relational databases. *Information Systems*, 104:101898, 2022.

[31] P.P. Chen. The Entity-Relationship Model – Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[32] James Rumbaugh, Ivar Jacobson, and Grady Booch. *Unified modeling language reference manual*. Pearson Higher Education, 2004.

[33] Arthur HM ter Hofstede, Ernst Lippe, and Paul J. M. Frederiks. Conceptual data modelling from a categorical perspective. *The Computer Journal*, 39(3):215–231, 1996.

[34] Jean-Jacques VR Wintraecken. *The NIAM information analysis method: theory and practice*. Springer Science & Business Media, 2012.

[35] Terry A Halpin and Maria E Orlowska. Fact-oriented modelling for data analysis. *Information Systems Journal*, 2(2):97–119, 1992.

[36] Terry Halpin. Fact-oriented modeling: Past, present and future. In *Conceptual modelling in information systems engineering*, pages 19–38. Springer, 2007.

[37] Richard Hull and Roger King. Semantic database modeling: Survey, applications, and research issues. *ACM Computing Surveys (CSUR)*, 19(3):201–260, 1987.

[38] L Kerschberg and JES Pacheco. A Functional Data Base Model, Technical Report, 1976.

[39] David W Shipman. The functional data model and the data languages DAPLEX. *ACM Transactions on Database Systems (TODS)*, 6(1):140–173, 1981.

[40] Michael Hammer and Dennis Mc Leod. Database description with SDM: A semantic database model. *ACM Transactions on Database Systems (TODS)*, 6(3):351–386, 1981.

[41] Serge Abiteboul and Richard Hull. IFO: A formal semantic database model. *ACM Transactions on Database Systems (TODS)*, 12(4):525–565, 1987.

[42] Jaroslav Pokorný and Karel Richta. Towards Conceptual and Logical Modelling of NoSQL Databases. In Emilio Insfran, Fernando González, Silvia Abrahão, Marta Fernández, Chris Barry, Michael Lang, Henry Linger, and Christoph Schneider, editors, *Advances in Information Systems Development*, pages 255–272, Cham, 2022. Springer International Publishing.

[43] Il-Yeol Song, Mary Evans, and Eun K Park. A comparative analysis of entity-relationship diagrams. *Journal of Computer and Software Engineering*, 3(4):427–459, 1995.

[44] David S. Reiner, Michael L. Brodie, Gretchen Brown, Mark Friedell, David Kramlich, John Lehman, and Arnon Rosenthal. The Database Design and Evaluation Workbench (DDEW) Project at CCA. *IEEE Database Eng. Bull.*, 7(4):10–15, 1984.

[45] Toby J Teorey. *Database modeling and design*. Morgan Kaufmann, 1999.

[46] Jeff Hoffer, Ramesh Venkataraman, and Heikki Topi. *Modern database management*. Pearson Education Limited, 2016.

[47] Thomas A Bruce. *Designing quality databases with IDEF1X information models*. Dorset House Publishing Co., Inc., 1992.

[48] Object Management Group. Object Constraint Language (OCL), version 2.4, 2014.

[49] Martin Necasky. XSEM: A Conceptual Model for XML. In *Proc. of APCCM '07 - Volume 67*, page 37–48, Aus, 2007. ACS, Inc.

[50] Marc Gyssens, Jan Paredaens, and Dirk Van Gucht. A graph-oriented object model for database end-user interfaces. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of data*, pages 24–33, 1990.

[51] Tomáš Hruška and Petr Kolenčík. Comparison of categorical foundations of object-oriented database model. In *International Conference on Deductive and Object-Oriented Databases*, pages 302–319. Springer, 1997.

[52] Zinovy Diskin and Boris Cadish. Variable sets and functions framework for conceptual modeling: Integrating ER and OO via sketches with dynamic markers. In *International Conference on Conceptual Modeling*, pages 226–237. Springer, 1995.

[53] Zinovy Diskin. Generalized sketches as an algebraic graph-based framework for semantic modeling and database design. *University of Latvia: Riga, Latvia*, 1997.

[54] Kristopher S Brown, David I Spivak, and Ryan Wisnesky. Categorical data integration for computational science. *Computational Materials Science*, 164:127–132, 2019.

[55] David I. Spivak. Functorial data migration. *Information and Computation*, 217:31–51, 2012.

[56] Zhen Hua Liu, Jiaheng Lu, Dieter Gawlick, Heli Helskyaho, Gregory Pogossiants, and Zhe Wu. Multi-model Database Management Systems - A Look Forward. In *VLDB '18 Workshops*, pages 16–29. Springer, 2019.

[57] Valter Uotila and Jiaheng Lu. A formal category theoretical framework for multi-model data transformations. In El Kindi Rezig, Vijay Gadepally, Timothy G. Mattson, Michael Stonebraker, Tim Kraska, Fusheng Wang, Gang Luo, Jun Kong, and Alevtina Dubovitskaya, editors, *Heterogeneous Data Management, Polystores, and Analytics for Healthcare - VLDB Workshops, Poly 2021 and DMAH 2021, Virtual Event, August 20, 2021, Revised Selected Papers*, volume 12921 of *Lecture Notes in Computer Science*, pages 14–28. Springer, 2021.

[58] Patrick Schultz, David I. Spivak, and Ryan Wisnesky. Algebraic Model Management: A Survey. In Phillip James and Markus Roggenbach, editors, *Recent Trends in Algebraic Development Techniques*, pages 56–69, Cham, 2017. Springer International Publishing.

[59] Ralph Abraham, Jerrold E Marsden, and Tudor Ratiu. *Manifolds, tensor analysis, and applications*, volume 75. Springer Science & Business Media, 2012.

[60] Alberto Hernández Chillón, Diego Sevilla Ruiz, and Jesús García Molina. Athena: A database-independent schema definition language. In Iris Reinhartz-Berger and Shazia W. Sadiq, editors, *Advances in Conceptual Modeling - ER 2021 Workshops CoMoNoS, EmpER, CMLS, St. John's, NL, Canada, October 18-21, 2021, Proceedings*, volume 13012 of *Lecture Notes in Computer Science*, pages 33–42. Springer, 2021.

[61] Kristóf Marussy, Oszkár Semeráth, Aren A Babikian, and Dániel Varró. A Specification Language for Consistent Model Generation based on Partial Models. *J. Object Technol.*, 19(3):3–1, 2020.

[62] Franz Baader, Diego Calvanese, Deborah McGuinness, Peter Patel-Schneider, Daniele Nardi, et al. *The description logic handbook: Theory, implementation and applications*. Cambridge university press, 2003.

[63] Pavel Contos. Abstract model for multi-model data. In Christian S. Jensen, Ee-Peng Lim, De-Nian Yang, Wang-Chien Lee, Vincent S. Tseng, Vana Kalogeraki, Jen-Wei Huang, and Chih-Ya Shen, editors, *Database Systems for Advanced Applications - 26th International Conference, DASFAA 2021, Taipei, Taiwan, April 11-14, 2021, Proceedings, Part III*, volume 12683 of *Lecture Notes in Computer Science*, pages 647–651. Springer, 2021.

[64] Pavel Koupil, Martin Svoboda, and Irena Holubova. MM-cat: A Tool for Modeling and Transformation of Multi-Model Data using Category Theory. In *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, 2021, pages 635–639, New York, NY, USA, 2021. IEEE.

[65] Mohamed-Amine Baazizi, Dario Colazzo, Giorgio Ghelli, and Carlo Sartiani. Parametric schema inference for massive JSON datasets. *The VLDB Journal*, 2019.

[66] D. Beckett. *RDF/XML Syntax Specification (Revised)*. W3c, 2004.

[67] Hanâ Lbath, Angela Bonifati, and Russ Harmer. Schema Inference for Property Graphs. In Yannis Velegrakis, Demetris Zeinalipour-Yazti, Panos K. Chrysanthis, and Francesco Guerra, editors, *Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021, Nicosia, Cyprus, March 23 - 26, 2021*, pages 499–504. OpenProceedings.org, 2021.

[68] Enrico Gallinucci, Matteo Golfarelli, Stefano Rizzi, Alberto Abelló, and Oscar Romero. Interactive multidimensional modeling of linked data for exploratory OLAP. *Inf. Syst.*, 77:86–104, 2018.

[69] Redouane Bouhamoum, Kenza Kellou-Menouer, Stephane Lopes, and Zoubida Kedad. Scaling up Schema Discovery for RDF Datasets. In *2018 IEEE Icdew*, pages 84–89. IEEE, 2018.

[70] Michael DiScala and Daniel J Abadi. Automatic Generation of Normalized Relational Schemas from Nested Key-Value Data. In *Sigmod '16*, pages 295–310, 2016.

[71] W3c. Extensible Markup Language (XML) 1.0 (Fifth Edition), 2008.

[72] Irena Mlýnková and Martin Nečaský. Heuristic Methods for Inference of XML Schemas: Lessons Learned and Open Issues. *Informatica*, 24(4):577–602, 2013.

[73] Geert Jan Bex, Frank Neven, Thomas Schwentick, and Stijn Vansummeren. Inference of Concise Regular Expressions and DTDs. *ACM Trans. Database Syst.*, 35(2):11:1–11:47, 2010.

[74] K. E. Shafer. Creating DTDs via the GB-Engine and Fred. In *Sgml'95*, page 399. Graphic Communications Association, 1995. http://xml.coverpages.org/shaferGB.html.

[75] Chuang-Hue Moh, Ee-Peng Lim, and Wee-Keong Ng. Re-engineering Structures from Web Documents. In *Proceedings of the fifth ACM Conference on Digital libraries*, Dl '00, pages 67–76, San Antonio, Texas, United States, 2000. ACM, New York, NY, USA.

[76] R. K. Wong and J. Sankey. *On Structural Inference for XML Data*. Report UNSW-CSE-TR-0313, School of Computer Science, The University of New South Wales, 2003.

[77] M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. XTRACT: a System for Extracting Document Type Descriptors from XML Documents. *SIGMOD Rec.*, 29:165–176, 2000.

[78] Ondřej Vošta, Irena Mlýnková, and Jaroslav Pokorný. Even an Ant Can Create an XSD. In *Database Systems for Advanced Applications*, volume 4947 of *Lecture Notes in Computer Science*, pages 35–50. Springer, 2008.

[79] Boris Chidlovskii. Schema Extraction from XML Collections. In *Proceedings of the 2nd ACM/IEEE-CS Joint Conference on Digital libraries*, Jcdl '02, pages 291–292, Portland, Oregon, USA, 2002. ACM, New York, NY, USA.

[80] H. Ahonen. *Generating Grammars for Structured Documents Using Grammatical Inference Methods*. Report A-1996-4, Department of Computer Science, University of Helsinki, 1996.

[81] Henning Fernau. Learning XML Grammars. In Petra Perner, editor, *Machine Learning and Data Mining in Pattern Recognition*, volume 2123 of *Lecture Notes in Computer Science*, pages 73–87. Springer, 2001.

[82] Jun-Ki Min, Jae-Yong Ahn, and Chin-Wan Chung. Efficient Extraction of Schemas for XML Documents. *Inf. Process. Lett.*, 85:7–12, 2003.

[83] Geert Jan Bex, Frank Neven, Thomas Schwentick, and Karl Tuyls. Inference of Concise DTDs from XML Data. In *Proceedings of the 32nd International Conference on Very Large Data Bases*, Vldb '06, pages 115–126, Seoul, Korea, 2006. VLDB Endowment.

[84] Geert Jan Bex, Wouter Gelade, Frank Neven, and Stijn Vansummeren. Learning Deterministic Regular Expressions for the Inference of Schemas from XML Data. *ACM Transactions on the Web*, 4(4):14:1–14:32, 2010.

[85] G. J. Bex, F. Neven, T. Schwentick, and S. Vansummeren. Inference of Concise Regular Expressions and DTDs. *ACM Trans. Database Syst.*, 35(2):11:1–11:47, 2010.

[86] Geert Jan Bex, Frank Neven, and Stijn Vansummeren. Inferring XML Schema Definitions from XML Data. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, Vldb '07, pages 998–1009, Vienna, Austria, 2007. VLDB Endowment.

[87] E. M. Gold. Language Identification in the Limit. *Information and Control*, 10(5):447–474, 1967.

[88] Julie Vyhnanovská and Irena Mlýnková. Interactive Inference of XML Schemas. In *Research Challenges in Information Science (RCIS), 2010 Fourth International Conference on*, pages 191–202. IEEE Computer Society Press, May 2010.

[89] Angelo Augusto Frozza, Ronaldo dos Santos Mello, and Felipe de Souza da Costa. An Approach for Schema Extraction of JSON and Extended JSON Document Collections. In *Iri 2018*, pages 356–363. IEEE, 2018.

[90] Severino Feliciano Morales. *Inferring NoSQL Data Schemas with Model-Driven Engineering Techniques*. PhD thesis, University of Murcia, Murcia, Spain, March 2017.

[91] Ivan Veinhardt Latták. and Pavel Koupil. A Comparative Analysis of JSON Schema Inference Algorithms. In *Proceedings of the 17th International Conference on Evaluation of Novel Approaches to Software Engineering - ENASE,*, pages 379–386. Insticc, SciTePress, 2022.

[92] Diego Sevilla Ruiz, Severino Feliciano Morales, and Jesús García Molina. Inferring Versioned Schemas from NoSQL Databases and Its Applications. In *Conceptual Modeling*, pages 467–480. Springer, 2015.

[93] Alberto Hernández Chillón, Severino Feliciano Morales, Diego Sevilla, and Jesús García Molina. Exploring the Visualization of Schemas for Aggregate-Oriented NoSQL Databases. In *Proceedings of the ER Forum 2017 and the ER 2017 Demo Track co-located with the 36th International Conference on Conceptual Modelling (ER 2017), Valencia, Spain, - November 6-9, 2017.*, pages 72–85, 2017.

[94] Meike Klettke, Uta Störl, and Stefanie Scherzinger. Schema Extraction and Structural Outlier Detection for JSON-based NoSQL Data Stores. In *Datenbanksysteme für Business, Technologie und Web (BTW), 16. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" (DBIS), 4.-6.3.2015 in Hamburg, Germany. Proceedings*, pages 425–444, 2015.

[95] Meike Klettke, Hannes Awolin, Uta Storl, Daniel Muller, and Stefanie Scherzinger. Uncovering the Evolution History of Data Lakes. In *2017 IEEE International Conference on Big Data*, pages 2380–2389, New York, United States, 2017. IEEE.

[96] Mark Lukas Möller, Nicolas Berton, Meike Klettke, Stefanie Scherzinger, and Uta Störl. jhound: Large-scale profiling of open JSON data. *Btw 2019*, 2019.

[97] Michael Fruth, Kai Dauberschmidt, and Stefanie Scherzinger. Josch: Managing Schemas for NoSQL Document Stores. In *Icde '21*, pages 2693–2696. IEEE, 2021.

[98] Javier Luis Cánovas Izquierdo and Jordi Cabot. Discovering Implicit Schemas in JSON Data. In *Icwe '13*, pages 68–83. Springer, 2013.

[99] Javier Luis Cánovas Izquierdo and Jordi Cabot. JSONDiscoverer: Visualizing the Schema Lurking behind JSON Documents. *Knowledge-Based Systems*, 103:52–55, 2016.

[100] Lanjun Wang, Shuo Zhang, Juwei Shi, Limei Jiao, Oktie Hassanzadeh, Jia Zou, and Chen Wangz. Schema Management for Document Stores. *Proc. VLDB Endow.*, 8(9):922–933, 2015.

[101] Felipe Pezoa, Juan L. Reutter, Fernando Suarez, Martín Ugarte, and Domagoj Vrgoč. Foundations of JSON Schema. In *Proceedings of the 25th International Conference on World Wide Web*, page 263–273, 2016.

[102] Mohamed-Amine Baazizi, Dario Colazzo, Giorgio Ghelli, and Carlo Sartiani. A Type System for Interactive JSON Schema Inference. In *Icalp 2019*, volume 132 of *LIPIcs*, pages 101:1–101:13, 2019.

[103] Angelo Augusto Frozza, Eduardo Dias Defreyn, and Ronaldo dos Santos Mello. A Process for Inference of Columnar NoSQL Database Schemas. In *Anais do XXXV Simpósio Brasileiro de Bancos de Dados*, pages 175–180. Sbc, 2020.

[104] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. XML Schema Part 1: Structures Second Edition. World Wide Web Consortium, Recommendation REC-xmlschema-1-20041028, October 2004.

[105] Paul V. Biron and Ashok Malhotra. XML Schema Part 2: Datatypes Second Edition, W3C Recommendation, October 2004. http://www.w3.org/TR/xmlschema-2/.

[106] JSON Schema – Specification, 2021.

[107] Ivan Veinhardt Latták. *Schema Inference for NoSQL Databases*. Master thesis, Charles University in Prague, Czech Republic, 2021.

[108] Burton H. Bloom. Space/Time Trade-Offs in Hash Coding with Allowable Errors. *Commun. ACM*, 13(7):422–426, 1970.

[109] Pavel Koupil, Sebastián Hricko, and Irena Holubová. MM-infer: A Tool for Inference of Multi-Model Schemas. In Julia Stoyanovich, Jens Teubner, Paolo Guagliardo, Milos Nikolic, Andreas Pieris, Jan Mühlig, Fatma Özcan, Sebastian Schelter, H. V. Jagadish, and Meihui Zhang, editors, *Proceedings of the 25th International Conference on Extending Database Technology, EDBT 2022, Edinburgh, UK, March 29 - April 1, 2022*, pages 2:566–2:569. OpenProceedings.org, 2022.

[110] Uta Störl and Meike Klettke. Darwin: A Data Platform for NoSQL Schema Evolution Management and Data Migration. In *Workshop Proceedings of the EDBT/ICDT 2022 Joint Conference (March 29-April 1, 2022), Edinburgh, UK*, 2022.

[111] Mark Lukas Möller, Meike Klettke, and Uta Störl. EvoBench - A Framework for Benchmarking Schema Evolution in NoSQL. In Xintao Wu, Chris Jermaine, Li Xiong, Xiaohua Hu, Olivera Kotevska, Siyuan Lu, Weija Xu, Srinivas Aluru, Chengxiang Zhai, Eyhab Al-Masri, Zhiyuan Chen, and Jeff Saltz, editors, *2020 IEEE International Conference on Big Data (IEEE BigData 2020), Atlanta, GA, USA, December 10-13, 2020*, pages 1974–1984. IEEE, 2020.

[112] Jiaheng Lu. Towards Benchmarking Multi-Model Databases. In *CIDR 2017*. www.cidrdb.org, 2017.

[113] Chao Zhang and Jiaheng Lu. Holistic evaluation in multi-model databases benchmarking. *Distributed and Parallel Databases*, 39:1–33, 2019.

[114] Carlo A. Curino, Hyun Jin Moon, Alin Deutsch, and Carlo Zaniolo. Update Rewriting and Integrity Constraint Maintenance in a Schema Evolution Support System: PRISM++. *Proc. VLDB Endow.*, 4(2):117–128, nov 2010.

[115] Souvik Bhattacherjee, Gang Liao, Michael Hicks, and Daniel J. Abadi. BullFrog: Online Schema Evolution via Lazy Evaluation. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21, page 194–206, New York, NY, USA, 2021. Association for Computing Machinery.

[116] Alberto Hernández Chillón, Meike Klettke, Diego Sevilla Ruiz, and Jesús García Molina. A Taxonomy of Schema Changes for NoSQL Databases. *arXiv preprint arXiv:2205.11660*, 2022.

[117] Marek Polák, Martin Nečaský, and Irena Holubová. DaemonX: Design, Adaptation, Evolution, and Management of Native XML (and More Other) Formats. In *Iiwas '13*, pages 484–493, New York, USA, 2013. Acm.

[118] Jakub Klímek, Jakub Malỳ, Martin Nečaskỳ, and Irena Holubová. eXolutio: methodology for design and evolution of XML schemas using conceptual modeling. *Informatica*, 26(3):453–472, 2015.

[119] M. Nečaský, J. Klímek, J. Malý, and I. Mlýnková. Evolution and Change Management of XML-based Systems. *Elsevier*, 85(3):683–707, 2012.

[120] Irena Holubová, Michal Vavrek, and Stefanie Scherzinger. Evolution Management in Multi-Model Databases. *Data Knowl. Eng.*, 136:101932, 2021.

[121] Jérôme Fink, Maxime Gobert, and Anthony Cleve. Adapting Queries to Database Schema Changes in Hybrid Polystores. In *20th IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2020, Adelaide, Australia, September 28 - October 2, 2020*, pages 127–131. IEEE, 2020.

[122] Andrea Hillenbrand, Maksym Levchenko, Uta Störl, Stefanie Scherzinger, and Meike Klettke. MigCast: Putting a Price Tag on Data Model Evolution in NoSQL Data Stores. In *Sigmod '19*, page 1925–1928. Acm, 2019.

[123] Steve Awodey. *Category Theory*. Oxford university press, 2010.

[124] David I Spivak. Category theory for scientists. *arXiv preprint arXiv:1302.6946*, 2013.

[125] Uta Störl, Daniel Müller, Alexander Tekleab, Stephane Tolale, Julian Stenzel, Meike Klettke, and Stefanie Scherzinger. Curating Variational Data in Application Development. In *Proc. of ICDE '18*, pages 1605–1608, 2018.

[126] Andrea Hillenbrand, Uta Störl, Maksym Levchenko, Shamil Nabiyev, and Meike Klettke. Towards Self-Adapting Data Migration in the Context of Schema Evolution in NoSQL Databases. In *ICDE Workshops 2020*, pages 133–138. IEEE, 2020.

[127] Stefanie Scherzinger, Meike Klettke, and Uta Störl. Managing schema evolution in NoSQL data stores. *arXiv preprint arXiv:1308.0514*, 2013.

[128] Andrea Hillenbrand, Uta Störl, Shamil Nabiyev, and Meike Klettke. Self-adapting data migration in the context of schema evolution in NoSQL databases. *Distributed and Parallel Databases*, 40(1):5–25, 2022.

[129] Andrea Hillenbrand, Stefanie Scherzinger, and Uta Störl. Remaining in Control of the Impact of Schema Evolution in NoSQL Databases. In *International Conference on Conceptual Modeling*, pages 149–159. Springer, 2021.

[130] Meike Klettke, Uta Störl, Manuel Shenavai, and Stefanie Scherzinger. NoSQL Schema Evolution and Big Data Migration at Scale. In *2016 IEEE International Conference on Big Data, BigData 2016, Washington DC, USA, December 5-8, 2016*, pages 2764–2774. IEEE, 2016.

[131] Uta Störl, Alexander Tekleab, Meike Klettke, and Stefanie Scherzinger. In for a Surprise When Migrating NoSQL Data. In *34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018*, page 1662. IEEE Computer Society, 2018.

[132] Mark Lukas Möller, Meike Klettke, Andrea Hillenbrand, and Uta Störl. Query Rewriting for Continuously Evolving NoSQL Databases. In *Conceptual Modeling - 38th International Conference, ER 2019, Salvador, Brazil, November 4-7, 2019, Proceedings*, volume 11788 of *Lncs*, pages 213–221. Springer, 2019.

[133] Irena Holubová, Pavel Koupil, and Jiaheng Lu. Self-Adapting Design and Maintenance of Multi-Model Databases, 2022.

[134] Brendan Fong and David I Spivak. Seven Sketches in Compositionality: An Invitation to Applied Category Theory, 2018.

[135] Michael Barr and Charles Wells. *Category Theory for Computing Science*, volume 1. Prentice Hall, New York, 1990.

List of Figures

0.1 An example of variety of data 8
0.2 An example of ER schema 14
0.3 An example of UML schema 16
0.4 An example of a conceptual schema (Lippe and Ter Hofstede) . . 18
0.5 An example of CGOOD application to multi-model data 21
0.6 An example of a logical categorical schema and data (Spivak et al.) 23
0.7 An example of APG (a) corresponding to data (b) 25
0.8 An example of (a) ETF and (b) EAO strategies 26
0.9 An example of Associative Arrays 28
0.10 An example of TDM 29
0.11 An example of U-Schema for graph (a) and document model (b) . 31
0.12 An example of a single addition (i.e., a pushout) 51
0.13 An example of schema mapping functor (a) inducing pullback (b), right pushforward (c) and left pushforward (d), i.e., data migration functors 53

A.1 An example of a category 97
A.2 An example of (not) a category 98
A.3 An example of a functor 99
A.4 An example of a set-valued functor (a) and a corresponding graph (b) 100
A.5 An example of commutative triangle (a) and square (b) 100
A.6 Naturality squares for *src* (a) and *tgt* (b) 102
A.7 An example of a natural transformation (a) and the corresponding graph homomorphism (b) 103
A.8 An example of a product corresponding to Cartesian product of sets 106
A.9 An example of a coproduct corresponding to disjoint union of sets 107
A.10 An example of a pullback corresponding to an intersection of sets 108
A.11 An example of a pushout corresponding to union of sets 108

List of Tables

0.1 Unification of terms in popular models 9
0.2 Expressive power of approaches modelling conceptual layer 19
0.3 Comparison of logical layer modeling approaches 32
0.4 Comparison of the selected schema inference approaches 46
0.5 A comparison of features in the selected evolution management approaches 57
0.6 A comparison and classification of supported SMOs in the selected evolution management approaches 57

A.1 Application of basic definitions 98
A.2 Application of functors 101
A.3 Application of natural transformation 103
A.4 Application of universal constructions 108

List of Abbreviations

AA Associative Arrays. 19, 32, 33

AI Artificial Intelligence. 10

APG Algebraic Property Graph. 22–25, 32, 87, 98, 101, 103, 108

BSON Binary JSON. 41–43, 45

CGOOD Categorical Graph-Oriented Object Data Model. 20, 21, 32, 87, 98, 101, 103, 108

CPER Categorical Path Equivalence Relation. 21

CQL Categorical Query Language. 21, 23, 32, 53

DBMS Database Management System. v, 5, 19, 50, 56, 59, 60, 62, 66, 68, 73

DML Data Manipulation Language. 62

DSL Domain Specific Language. 61

DTD Document Type Definition. 40, 48

EAO Input per Each Aggregated Object. 25, 26, 87

ER Entity-Relationship. 12–15, 18, 33, 34, 37, 38, 46, 48, 66, 68, 73, 87

ETF Input per Each Top-Level Property. 25, 26, 87

FDM Functional Data Model. 12

FORM Fact-Oriented Modelling. 12

GOOD Graph-Oriented Object Database Model. 20

IC Integrity Constraint. 46

ICMO Integrity Constraints Modification Operation. 59

IDEF1X Integration DEFinition for Information Modeling. 14

IFO IFO Model. 12

IRI Internationalized Resource Identifier. 9

JSON JavaScript Object Notation. 7, 9, 22, 25, 35, 39–48, 53, 58

MDE Model-Driven Engineering. 41, 46

MMSEL Multi-Model Schema Evolution Language. 55, 61

NIAM Natural language Information Analysis Method. 12

NoAM NoSQL Abstract Model. 19, 25, 27, 31–33

NoSQL Not only SQL. 5, 7, 19, 25, 39, 41, 44, 46, 50, 55, 59, 61, 66

OCL Object Constraint Language. 18, 34, 45

PBA Property-Based Algorithm. 49

PDF Property Domain Footprint. 49

PIM Platform-Independent Model. 12

PSM Platform-Specific Model. 18

RBA Record-Based Algorithm. 49

RDBMS Relational Database Management System. 5

RDF Resource Description Framework. 7, 9, 21, 22, 25, 32, 37, 39, 40, 53, 57

RSD Record Schema Description. 49

SDM Semantic Data Model. 12

SEL Schema Evolution Language. 54

SMO Schema Modification Operation. 50–52, 54–62, 73, 89, 101

SPARQL SPARQL Protocol and RDF Query Language. 74

SQL Structured Query Language. 7

TDM Tensor Data Model. 19, 28, 29, 32, 87

UML Unified Modeling Language. 12–16, 18, 33, 34, 38, 39, 43, 44, 46, 48, 68, 87

UMP Universal Mapping Property. 106

UP Universal Property. 106

XML eXtensible Markup Language. 7, 9, 19, 22, 25, 35, 39, 40, 46–48

List of Publications

Journal Paper I: Pavel Koupil, and Irena Holubová. A unified representation and transformation of multi-model data using category theory. *J Big Data* 9, 61 (2022). doi: 10.1186/s40537-022-00613-3 **(Q1, IF: 14.57, SJR: 2.592)**

Contribution: The share of the author's contributions in this paper is equal. All authors read and approved the final manuscript.

Journal Paper II: Pavel Koupil, Sebastián Hricko, and Irena Holubová. A Universal Approach for Multi-Model Schema Inference. *J Big Data* (accepted). doi: 10.1186/s40537-022-00645-9 **(Q1, IF: 14.57, SJR: 2.592)**

Contribution: The share of the author's contributions in this paper is equal. All authors read and approved the final manuscript.

Conference Paper I: Pavel Čontoš (Koupil), and Martin Svoboda. JSON schema inference approaches. *1st International Workshop on Conceptual Modeling for NoSQL Data Stores, CoMoNoS 2020.* Vienna, Austria, November 2020. doi: 10.1007/978-3-030-65847-2_16 **(Workshop@CORE A)**

Contribution: The share of the author's contributions in this paper is equal. All authors read and approved the final manuscript.

Conference Paper II: Pavel Čontoš (Koupil). Abstract Model for Multi-model Data. *26th International Conference on Database Systems for Advanced Applications, DASFAA 2021.* Taipei, Taiwan, April 2021. doi: 10.1007/978-3-030-73200-4_53 **(PhD Consortium@CORE B)**

Contribution: The share of the author's contributions in this paper is equal. All authors read and approved the final manuscript.

Conference Paper III: Martin Svoboda, Pavel Čontoš (Koupil), and Irena Holubová. Categorical Modeling of Multi-Model Data: One Model to Rule Them All. *10th International Conference on Model and Data Engineering, MEDI 2021.* Tallinn, Estonia, June 2021. doi: 10.1007/978-3-030-78428-7_15 **(CORE C)**

Contribution: The share of the author's contributions in this paper is equal. All authors read and approved the final manuscript.

Conference Paper IV: Irena Holubová, Pavel Čontoš (Koupil), and Martin Svoboda. Categorical Management of Multi-Model Data. *25th International Database Engineering & Applications Symposium, IDEAS 2021.* Montreal, Canada, July 2021. doi: 10.1145/3472163.3472166 **(CORE B)**

Contribution: The share of the author's contributions in this paper is equal. All authors read and approved the final manuscript.

Conference Paper V: Irena Holubová, Pavel Čontoš (Koupil), and Martin Svoboda. Multi-Model Data Modeling and Representation: State of the Art and Research Challenges. *25th International Database Engineering & Applications Symposium, IDEAS 2021.* Montreal, Canada, July 2021. doi: 10.1145/3472163.3472267 **(CORE B)**

Contribution: The share of the author's contributions in this paper is equal. All authors read and approved the final manuscript.

Conference Paper VI: Pavel Koupil, Martin Svoboda, and Irena Holubová. MM-cat: A Tool for Modeling and Transformation of Multi-Model Data using Category Theory. *24th International Conference on Model Driven Engineering Languages and Systems, MODELS 2021.* Fukuoka, Japan, October 2021. doi: 10.1109/MODELS-C53483.2021.00098 **(CORE A)**

Contribution: The share of the author's contributions in this paper is equal. All authors read and approved the final manuscript.

Conference Paper VII: Pavel Koupil, and Irena Holubová. Unifying Categorical Representation of Multi-Model Data. *37th ACM/SIGAPP Symposium On Applied Computing, SAC 2022.* Brno, Czech Republic, April 2022. doi: 10.1145/3477314.3507690 **(CORE B)**

Contribution: The share of the author's contributions in this paper is equal. All authors read and approved the final manuscript.

Conference Paper VIII: Ivan Veinhardt Latták, and Pavel Koupil. A Comparative Analysis of JSON Schema Inference Algorithms. *17th International Conference on Evaluation of Novel Approaches to Software Engineering, ENASE 2022.* Virtual Event, April 2022. doi: 10.5220/0011046000003176 **(CORE B)**

Contribution: The paper is based on Ivan Veinhardt Latták's Master thesis [107] (supervised by Pavel Koupil).

Conference Paper IX: Pavel Koupil, Sebastián Hricko, and Irena Holubová. MM-infer: A Tool for Inference of Multi-Model Schemas. *29th International Conference on Extending Database Technology, EDBT 2022.* Edinburgh, UK, March 2022. doi: 10.48786/edbt.2022.52 **(CORE A)**

Contribution: The share of the author's contributions in this paper is equal. All authors read and approved the final manuscript.

Conference Paper X: Irena Holubová, Pavel Koupil, and Jiaheng Lu. Self-Adapting Design and Maintenance of Multi-Model Databases. *26th International Database Engineering & Applications Symposium, IDEAS 2022.* (accepted) **(CORE B)**

Contribution: The share of the author's contributions in this paper is equal. All authors read and approved the final manuscript.

Conference Paper XI: Pavel Koupil, Sebastián Hricko, and Irena Holubová. Schema Inference for Multi-Model Data. *25th International Conference on Model Driven Engineering Languages and Systems, MODELS 2022.* (accepted) **(CORE A)**

Contribution: The share of the author's contributions in this paper is equal. All authors read and approved the final manuscript.

Conference Paper XII: Pavel Koupil, Jáchym Bártík, and Irena Holubová. *MM-evocat*: A Tool for Modelling and Evolution Management of Multi-Model Data. Manuscript under review.

Contribution: The share of the author's contributions in this paper is equal.

A. Category Theory

Category theory [17] is a branch of mathematics that provides a way to generalise mathematical structures and the relationships between them. Hence, it is a unifying theory that is useful for finding connections between different areas, not only in mathematics and theoretical computer science. We assume that by applying category theory we will achieve a reasonable level of abstraction of various data models and their combinations that will allow us to perform data migration, querying, and evolution management of multi-model data in a unified way.

In this section, we provide the basic definitions underlying our approach of multi-model schema and data representation (see Paper II), data migration (see Paper III), and schema and data evolution (see Paper V), as well as the approaches we have been inspired by [18, 19, 20, 21]. **Note that our aim is to propose an intuitive and user-friendly approach. Therefore, we explicitly use only the most basic constructs of category theory, i.e., primarily categories** (see Definition 1) **and functors** (see Definition 6). As for the other constructs, i.e., natural transformations (see Definition 10) and universal constructions (see Definitions 13, 14, 16, and 17), these are considered only implicitly, i.e., we do not require the user to have active knowledge of these definitions. On the contrary, these complex constructs are explicitly utilised in the approaches we are inspired by.

In addition, we also provide illustrative examples that are closely related to real-world applications in data modelling approaches at the conceptual and logical level. Finally, for the convenience of the reader, at the end of each subsection we outline in which approaches and for which purpose the above definitions are applied, i.e., we provide additional examples.

For more details we refer an interested reader to [134, 124, 123, 135, 17], which are ordered by difficulty, while for computer science we recommend particularly [134].

A.1 Basic Definitions

In this subsection, we introduce the basic definitions and concepts which form the foundation of category theory. Specifically, we introduce the notion of a category as a collection of objects and morphisms (sometimes called arrows), and we introduce different classes of morphisms based on their properties, as well as a way to create a category from an arbitrary graph. We conclude with illustrative examples of categories.

Definition 1. *The* ***category*** $\mathbf{C}$ *is a quadruple* $(\mathcal{O}_{\mathbf{C}}, \mathcal{M}_{\mathbf{C}}, \circ, 1)$ *such that:*

- $\mathcal{O}_{\mathbf{C}}$ *is a collection of* ***objects****.*
- $\mathcal{M}_{\mathbf{C}}$ *is a collection of* ***morphisms*** *where each* $f \in \mathcal{M}_{\mathbf{C}}$ *is represented as an arrow* $f : A \to B$ *(also denoted* $A \xrightarrow{f} B$*), where* $A, B \in \mathcal{O}_{\mathbf{C}}$*, such that* A *is the domain of* f*, denoted by* $dom(f) = A$*, and* B *is the codomain of* f*, denoted by* $cod(f) = B$*.*

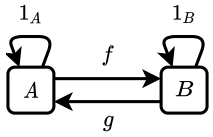
- *Given $f, g \in \mathcal{M}_{\mathbf{C}}$ and $cod(f) = dom(g)$, there exists $g \circ f \in \mathcal{M}_{\mathbf{C}}$, which we refer to as the* ***composition*** *of f and g. Moreover, the composition must be associative, i.e., for any $f, g, h \in \mathcal{M}_{\mathbf{C}}$ such that $cod(f) = dom(g)$ and $cod(g) = dom(h)$, the equality $h \circ (g \circ f) = (h \circ g) \circ f$ holds.*
- *For each object $A \in \mathcal{O}_{\mathbf{C}}$, there is exactly one* ***identity*** *morphism $1_A : A \to A$ such that $f \circ 1_A = f = 1_B \circ f$ holds for any $f : A \to B$, $f \in \mathcal{M}_{\mathbf{C}}$.*

Definition 2. *Let $\mathbf{C}$ be a category and $A, B \in \mathcal{O}_{\mathbf{C}}$. Then we define the* ***hom-class*** *$hom_{\mathbf{C}}(A, B) \subset \mathcal{M}_{\mathbf{C}}$ as the collection of all morphisms $f : A \to B$.*

Definition 3. *We call the category $\mathbf{C}$ a* ***small category*** *if both $\mathcal{O}_{\mathbf{C}}$ and $\mathcal{M}_{\mathbf{C}}$ are sets. Otherwise, we call the category $\mathbf{C}$ a* ***large category****. We say that $\mathbf{C}$ is* ***locally small category*** *if for any two objects $A, B \in \mathcal{O}_{\mathbf{C}}$ it holds that $hom_{\mathbf{C}}(A, B)$ forms a set.*

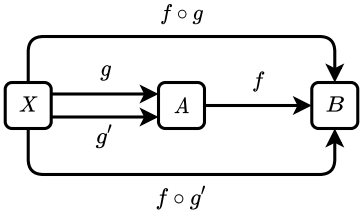
Definition 4. *Let $\mathbf{C}$ be a category, $A, B, X \in \mathcal{O}_{\mathbf{C}}$ and $f, g, g', g'' \in \mathcal{M}_{\mathbf{C}}$.*

A morphism $f : A \to B$ is an ***isomorphism*** *if and only if there exists a morphism $g : B \to A$ such that the composition of f and g yields identities, i.e., $g \circ f = 1_A$ and $f \circ g = 1_B$. Moreover, the morphism g is uniquely determined. That is, if there exist $g', g'' : B \to A$ such that $g' \circ f = 1_A$ and $f \circ g'' = 1_B$, then it must hold that $g' = g' \circ 1_B = g' \circ f \circ g'' = 1_A \circ g'' = g''$. We will denote the morphism g by f^{-1} and call it the* ***inverse morphism*** *of f.*



If there is a pair of isomorphisms $f : A \to B$, $g : B \to A$, then also A is isomorphic to B, which we denote by $A \cong B$. Note also that identity morphisms are trivial isomorphisms.

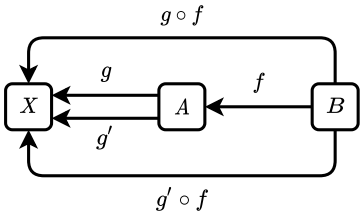
A morphism $f : A \to B$ is a ***mono****(morphism),*[1] *if for any object X and arbitrary two morphisms $g, g' : X \to A$ the following implication holds: $f \circ g = f \circ g' \implies g = g'$.*



A morphism $f : B \to A$ is an ***epi****(morphism),*[2] *if for any object X and arbitrary pair of morphisms $g, g' : A \to X$ the following implication holds: $g \circ f = g' \circ f \implies g = g'$.*

[1] A special case of a monomorphism is an injective morphism, but not every morphism is an injective morphism. Monomorphism is a more general notion than injection.

[2] A special case of epimorphism is a surjective morphism, but not every epimorphism is a surjective morphism. An epimorphism is a more general notion than surjection.



Definition 5. *Let $G = (V, E, src, tgt)$ be a graph such that V is the set of vertices, E is the set of edges, and $src : E \to V$, $tgt : E \to V$ are functions assigning the source vertex and the target vertex to an edge.*

The category $\mathbf{Free(G)} = \{\mathcal{O}_{\mathbf{Free(G)}}, \mathcal{M}_{\mathbf{Free(G)}}, \circ, 1\}$*, referred to as the* ***free category*** *on G, is the category with $\mathcal{O}_{\mathbf{Free(G)}}$ equal to V, $hom_{\mathbf{Free(G)}}(A, B)$, equal to all paths from v_a to v_b in G, such that $A, B \in \mathcal{O}_{\mathbf{Free(G)}}$ and $v_a, v_b \in V$, composition is determined by the concatenation of paths, and identity morphisms $1_A : A \to A$, $1_A \in \mathcal{M}_{\mathbf{Free(G)}}$ on an object is the trivial path at $v_a \in V$.*

Example A.1. The category **Set** is a category in which objects are sets and morphisms are functions between the sets. The composition of morphisms is given by the composition of functions, and the identity morphism is the identity function. Note also that the category **Set** has both initial and terminal objects. □

Example A.2. Figure A.1 illustrates a category $\mathbf{G} = (\mathcal{O}_{\mathbf{G}}, \mathcal{M}_{\mathbf{G}}, \circ_{\mathbf{G}}, 1_{\mathbf{G}})$, where $E, V \in \mathcal{O}_{\mathbf{G}}$ and $1_E, 1_V, src, tgt \in \mathcal{M}_{\mathbf{G}}$. The category $\mathbf{G}$ is indeed a category as the identity morphisms over all vertices are defined, i.e., 1_E and 1_V, and the associativity law for morphism composition holds.

Also note that the category $\mathbf{G}$ corresponds to the schema of an arbitrary directed graph $G = (V_G, E_G, src_G, tgt_G)$, i.e., the objects V and E correspond to the elements of the graph V_G and E_G and the morphisms src and tgt represent the functions src_G and tgt_G. We will show a particular graph in Example A.5 after we define the notion of a functor between categories. □

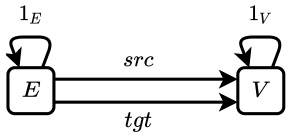


Figure A.1: An example of a category

Example A.3. Although a category is a special type of directed multi-graph, not every directed (multi-)graph is a category. Let the graph $G = (V, E, src, tgt)$ consist of vertices $A, B, C \in V$ and edges $f, g \in E$ (see Figure A.2 (a)). In this instance, it is not a category, e.g., because the graph does not contain a reflexive edge at any vertex that would correspond to an identity morphism.

Figure A.2 (b) illustrates the graph G', where $E' = E \cup \{1_A, 1_B, 1_C\}$. Again, this is not a category, as the path formed by the composition of the paths of the graph G', e.g., $g \circ f$, is not included in the graph.

Finally, Figure A.2 (c) illustrates the graph G'', where $E'' = E' \cup \{h\}$, $h = g \circ f$, which corresponds to the category. In other words, the graph G freely generates the category $\mathbf{Free(G)}$ (see Definition 5). □

Finally, Table A.1 summarises the application of categories in approaches representing data at the conceptual or logical level.

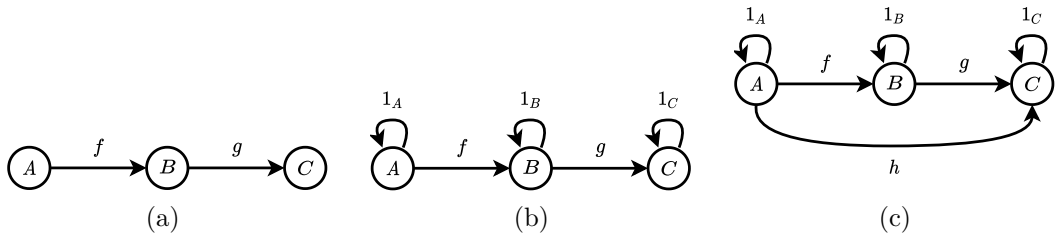


Figure A.2: An example of (not) a category

Table A.1: Application of basic definitions

	Lippe and Ter Hofstede [18]	**CGOOD** [19]	**Spivak et al.** [20]	**APG** [21]	**MM-(evo)cat** [3, 5]
Category (Definition 1)	Conceptual schema	Abstraction of schema	Schema	Abstraction of property labelled graph	Schema and instance category (early approach)
Small category (Definition 3)	-	-	Schema	-	Schema and instance category
Monomorphism (Definition 4)	Uniqueness	-	-	-	Uniqueness, inheritance
Epimorphism (Definition 4)	Simple and structured attributes	-	-	-	Simple and structured attributes
Free category (Definition 5)	-	-	Schema	-	Schema category

A.2 Functors

So far we have defined the elementary constructs of category theory. In this subsection, we add functors, i.e., structure-preserving mappings between categories that (not coincidentally) resemble morphisms between objects. Once again, we conclude with illustrative examples.

Definition 6. *Let* $\mathbf{C} = \{\mathcal{O}_\mathbf{C}, \mathcal{M}_\mathbf{C}, \circ, 1\}$ *and* $\mathbf{D} = \{\mathcal{O}_\mathbf{D}, \mathcal{M}_\mathbf{D}, \circ, 1\}$ *be categories. A* ***functor*** $F : \mathbf{C} \to \mathbf{D}$*, also denoted* $\mathbf{C} \xrightarrow{F} \mathbf{D}$*, is a structure-preserving mapping between categories that assigns objects* $\mathcal{O}_\mathbf{C}$ *to objects in* $\mathcal{O}_\mathbf{D}$ *and morphisms in* $\mathcal{M}_\mathbf{C}$ *to morphisms in* $\mathcal{M}_\mathbf{D}$*. For the functor* F*, the following holds:*

- $dom(F(f)) = F(dom(f))$ *and* $cod(F(f)) = F(cod(f))$ *for each morphism* $f \in \mathcal{M}_\mathbf{C}$*.*
- *Preserving of composition* $F(g \circ f) = F(g) \circ F(f)$ *for every pair of morphisms* $f, g \in \mathcal{M}_\mathbf{C}$ *such that* $dom(g) = cod(f)$*.*
- *Preserving of identity* $F(1_A) = 1_{F(A)}$ *for each object* $A \in \mathcal{O}_\mathbf{C}$*.*

Definition 7. *Let* $\mathbf{C}$ *and* $\mathbf{D}$ *be categories and* $F : \mathbf{C} \to \mathbf{D}$ *and* $G : \mathbf{D} \to \mathbf{E}$ *be functors. The* ***composition of the functors*** F *and* G *is a functor* $G \circ F : \mathbf{C} \to \mathbf{E}$ *such that:*

- *For every object* $A \in \mathcal{O}_{\mathbf{C}}$ *it holds that* $G \circ F(A) = G(F(A))$.
- *For every morphism* $f : A \to B$, $f \in \mathcal{M}_{\mathbf{C}}$ *it holds that* $G \circ F(f) = G(F(f))$.

Definition 8. *Let* $\mathbf{C}$ *and* $\mathbf{J}$ *be categories. The diagram of the form* $\mathbf{J}$ *in the category* $\mathbf{C}$ *is a functor* $D : \mathbf{J} \to \mathbf{C}$. *We refer to* $\mathbf{J}$ *as the index category of the diagram* D.

Definition 9. *We say that a* ***diagram commutes*** *if every two paths* $p = f_m \circ ... \circ f_1$ *and* $q = g_n \circ ... \circ g_1$, *where* $m, n \in \mathbb{N}$, $f_1, \ldots, f_m, g_1, \ldots, g_m, p, q \in \mathcal{M}_{\mathbf{C}}$, $dom(p) = dom(q)$, *and* $cod(p) = cod(q)$, *determine the same morphism via composition, i.e.,* $p = q$.

Example A.4. Figure A.3 illustrates examples of functors $F : \mathbf{C} \to \mathbf{D}$ (depicted in blue). Figure A.3 (a) corresponds to usual expectation of how functor $F : \mathbf{C} \to \mathbf{D}$ should map objects between two categories. Figure A.3 (b) illustrates functor F mapping objects A, B to B' and the morphism f to the identity morphism $1_{B'}$. Figure A.3 (c) maps object A to A' and B to C' with morphism f being mapped to the composition of morphisms $g' \circ f'$. In contrast, Figure A.3 (d) does not represent a functor because (1) object A is mapped to more than one object and (2) object B is not mapped at all. Figure A.3 (e) does not illustrate the functor either, since the structure of the category $\mathbf{C}$ is not preserved, i.e., the morphism f cannot be mapped. □

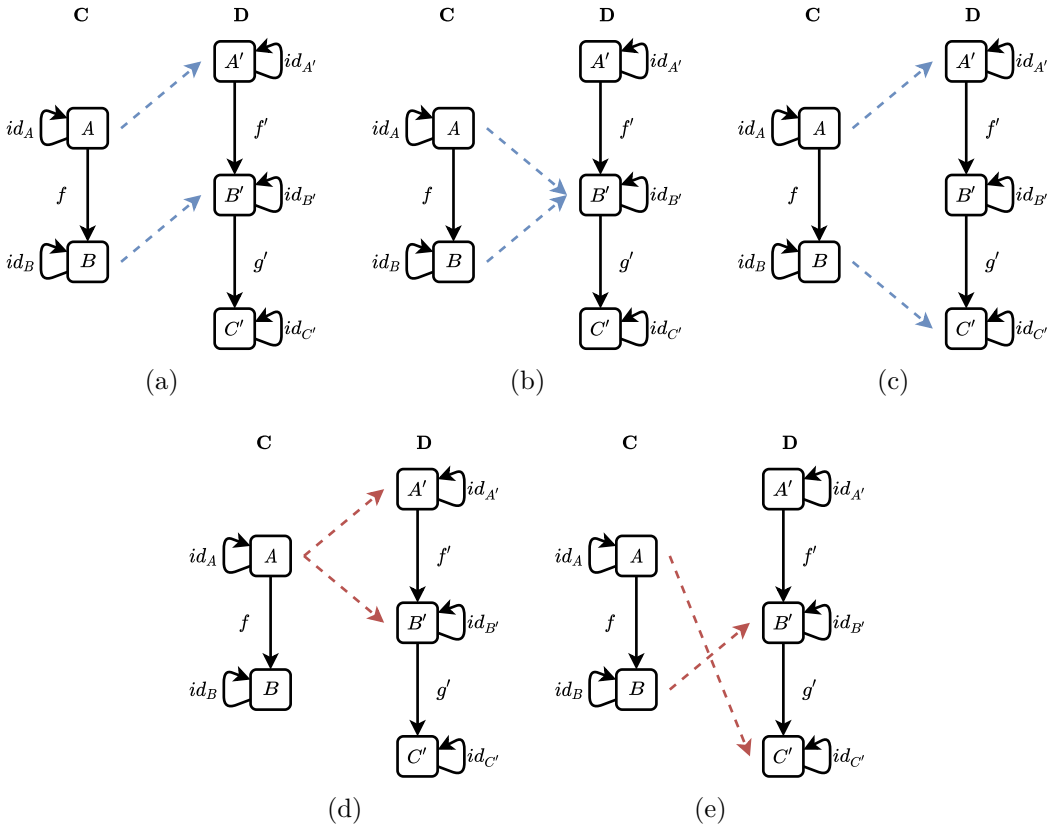
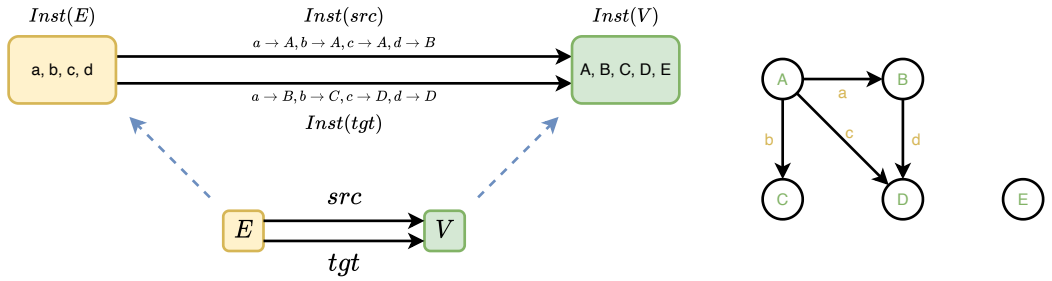


Figure A.3: An example of a functor

Example A.5. An example of a functor is the *set-valued functor* $Inst : \mathbf{G} \to \mathbf{Set}$, where $\mathbf{G}$ is the category from Example A.2 and $\mathbf{Set}$ is the category of all sets from Example A.1. It maps each object of category $\mathbf{G}$ to an object of category $\mathbf{Set}$, i.e., it assigns the object E to the set (of edges) $Inst(E) = E_{\mathbf{Set}}$ and the object V to the set (of vertices) $Inst(V) = V_{\mathbf{Set}}$ (see Figure A.4 (a)). The corresponding graph, represented by the (one of many) functor $Inst : \mathbf{G} \to \mathbf{Set}$, is depicted in Figure A.4 (b). (Note that for clarity only, we do not depict identity morphisms in the figure.) □

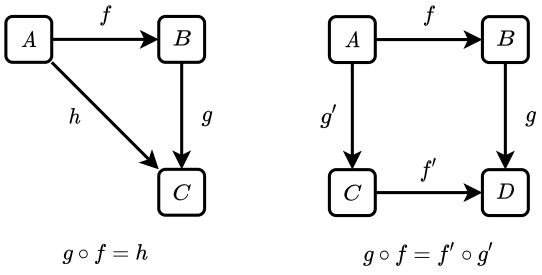


(a) (b)

Figure A.4: An example of a set-valued functor (a) and a corresponding graph (b)

Example A.6. The commutativity of the diagram depicted in Figure A.5 (a) implies that $g \circ f = h$, i.e., $dom(g \circ f) = dom(h)$ and $cod(g \circ f) = cod(h)$ and both $g \circ f$ and h lead to the same result. Note that this commutative diagram is referred to as a *commutative triangle*.

Similarly, the commutativity of the diagram depicted in Figure A.5 (b) implies that $g \circ f = f' \circ g'$, where $dom(g \circ f) = dom(f' \circ g')$ and $cod(g \circ f) = cod(f' \circ g')$. The commutative diagram is referred to as a *commutative square*. □



(a) (b)

Figure A.5: An example of commutative triangle (a) and square (b)

To conclude, Table A.2 summarises the application of functors in approaches representing data at the conceptual or logical level and in approaches performing data migration.

Table A.2: Application of functors

	Lippe and Ter Hofstede [18]	**CGOOD** [19]	**Spivak et al.** [20]	**APG** [21]	**MM-(evo)cat** [3, 5]
Functor (Definition 6)	-	Particular schema, data, and data instance	Particular data instance; SMOs; data migration	Particular property labelled graph	Particular data instance (evolution approach); SMOs; data migration

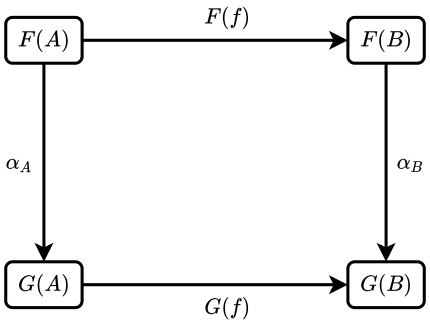
A.3 Natural Transformations

In this subsection, we define natural transformations that allow us to have multiple views of the same concept using different levels of abstraction. In addition, we define vertical composition of natural transformations, natural isomorphism, and functor category, where the objects of this category are functors and the morphisms are natural transformations. We conclude with illustrative examples of universal transformations and a functor category.

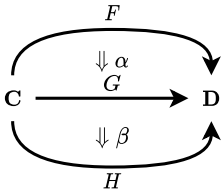
Definition 10. *Let* $\mathbf{C}$ *and* $\mathbf{D}$ *be categories and let* $F : \mathbf{C} \to \mathbf{D}$ *and* $G : \mathbf{C} \to \mathbf{D}$ *be functors.*

The ***natural transformation*** α *from* F *to* G*, denoted* $\alpha : F \to G$*, is a collection of components (i.e., morphisms)* α_A*,* $A \in \mathcal{O}_{\mathbf{C}}$*, that satisfy the commutative square law, as follows:*

- *For each object* $A \in \mathcal{O}_{\mathbf{C}}$ *there is a morphism* $\alpha_A : F(A) \to G(A)$*,* $\alpha_A \in \mathcal{M}_{\mathbf{D}}$*, called the* A***-component*** *of the natural transformation* α*.*
- *For each morphism* $f : A \to B$*,* $f \in \mathcal{M}_{\mathbf{C}}$*,* $A, B \in \mathcal{O}_{\mathbf{C}}$*, the following square, called* ***naturality square****, commutes, i.e.,* $\alpha_B \circ F(f) = G(f) \circ \alpha_A$*.*



Definition 11. *Let* $\mathbf{C}$ *and* $\mathbf{D}$ *be categories, let* $F, G, H : \mathbf{C} \to \mathbf{D}$ *be functors, and let* $\alpha : F \to G$ *and* $\beta : G \to H$ *be natural transformations.*



Vertical composition *of natural transformations* $\beta \circ \alpha$ *is a composition in which for each object* $c \in \mathcal{O}_{\mathbf{C}}$ *there exists a morphism* $(\beta \circ \alpha)_c \in \mathcal{M}_{\mathbf{D}}$ *for which* $(\beta \circ \alpha)_c = \beta_c \circ \alpha_c$ *(i.e., it commutes).*

Definition 12. *Let* $\mathbf{C}$ *and* $\mathbf{D}$ *be categories and let* $F : \mathbf{C} \to \mathbf{D}$ *and* $G : \mathbf{C} \to \mathbf{D}$ *be functors. A natural transformation* $\alpha : F \to G$ *is said to be a* ***natural isomorphism*** *if every component of* $\alpha_A : F(A) \to G(A)$ *is an isomorphism. In that case, the functors* F *and* G *are called* ***naturally isomorphic****.*

Lemma 1. *Let* $\mathbf{C}$ *and* $\mathbf{D}$ *be categories,* $F, G, H : \mathbf{C} \to \mathbf{D}$ *functors,* $\alpha : F \to G$ *and* $\alpha' : G \to H$ *natural transformations, and* $1_F : F \to F$ *a natural isomorphism.*

There exists a ***functor category***[3]*, denoted* $\mathbf{D}^{\mathbf{C}}$*, such that:*

- $\mathcal{O}_{\mathbf{D}^{\mathbf{C}}}$ *is a collection of functors* $F : \mathbf{C} \to \mathbf{D}$*.*
- $\mathcal{M}_{\mathbf{D}^{\mathbf{C}}}$ *is a collection of natural transformations* $\alpha : F \to G$*.*
- *The composition* $\alpha' \circ \alpha$ *is a vertical composition of natural transformations.*
- *The identity* 1_F *on the object* F *is a natural isomorphism.*

Proof. See [123]. □

Example A.7. Let $\mathbf{G}$ be the category from Example A.2, **Set** be the category from Example A.1 and let $Inst, Inst' : \mathbf{G} \to \mathbf{Set}$ be functors from Example A.5.

The natural transformation $\alpha : Inst \to Inst'$ involves two components, $\alpha_E : Inst(E) \to Inst'(E)$ and $\alpha_V : Inst(V) \to Inst'(V)$ and two naturality squares, $\alpha_V \circ Inst(src) = Inst'(src) \circ \alpha_E$ (see Figure A.6 (a)) and $\alpha_V \circ Inst(tgt) = Inst'(tgt) \circ \alpha_E$ (see Figure A.6 (b)).

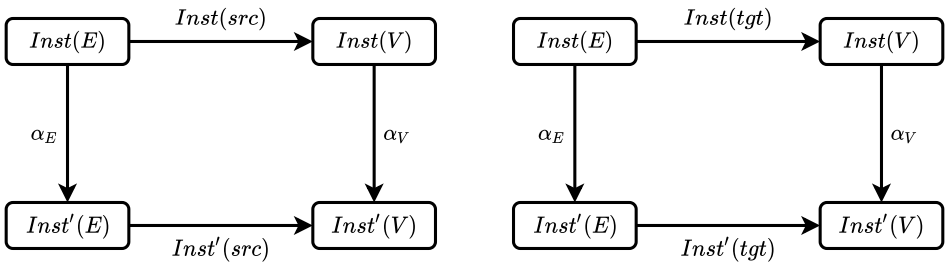


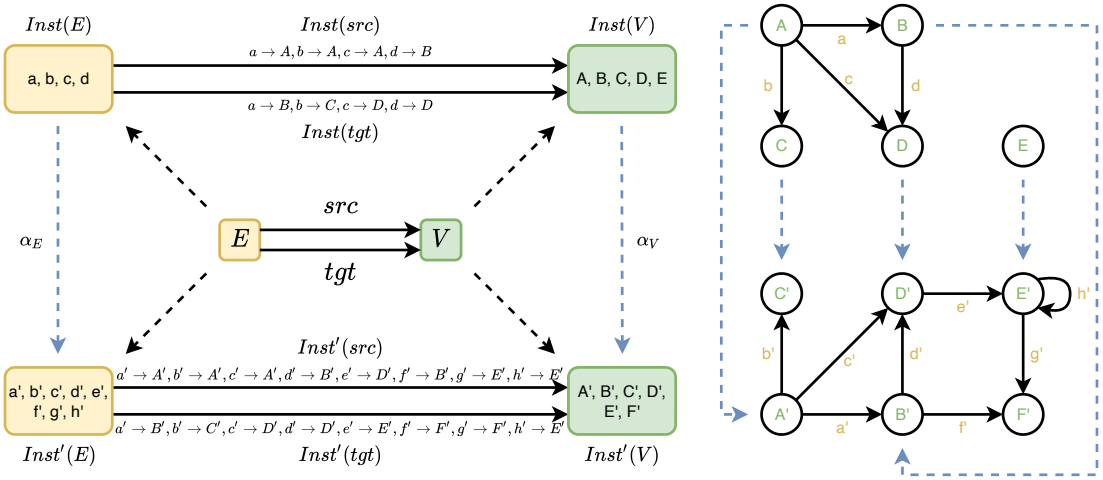
Figure A.6: Naturality squares for src (a) and tgt (b)

In other words, the natural transformation $\alpha : Inst \to Inst'$ (see Figure A.7 (a)) is the same as the graph homomorphism [124] (see Figure A.7 (b)).

□

Example A.8. Let $\mathbf{G}$ be the category from Example A.2, **Set** be the category of all sets from Example A.1, let $Inst, Inst', Inst'' : \mathbf{G} \to \mathbf{Set}$ be functors from

[3]Note that there exists also horizontal composition of natural transformations and a category, where the objects are categories and the morphisms are horizontal natural transformations of functors [123].



(a) (b)

Figure A.7: An example of a natural transformation (a) and the corresponding graph homomorphism (b)

Example A.5 (i.e., particular graphs) and let $\alpha : Inst \to Inst'$ a $\beta : Inst' \to Inst''$ be natural transformations from Example A.7 (i.e., graph homomorphisms).

The category of all graphs is the functor category $\mathbf{Set}^{\mathbf{G}}$ such that the objects of this category are all graphs and the morphisms are graph homomorphisms. □

Finally, Table A.3 summarises the application of natural transformation in approaches representing data at the conceptual or logical level and in approaches performing data migration.

Table A.3: Application of natural transformation

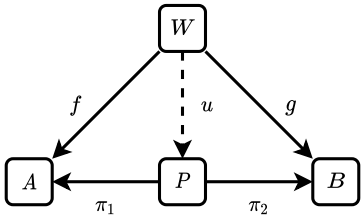
	Lippe and Ter Hofstede [18]	**CGOOD** [19]	**Spivak et al.** [20]	**APG** [21]	**MM-(evo)cat** [3, 5]
Natural transformation (Definition 10)	-	Allowed operations between schemas	Allowed operations between data instances conforming to the same schema	Allowed operations between property labelled graphs	Allowed operations between data instances conforming to the same schema (evolution approach)
Functor category (Lemma 1)	-	Category of all schemas	Category of all instances conforming to the same schema	Category of all property labelled graphs	Category of all instances conforming to the same schema (evolution approach)

A.4 Universal Constructions

In this subsection, we define so-called *universal constructions* that may resemble notions from set theory, such as, e.g., Cartesian product, disjunctive union,

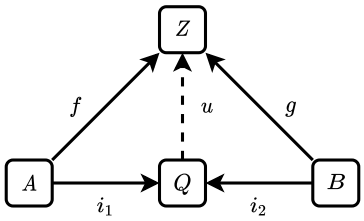
intersection, and union. We then conclude with examples to show that this similarity is not coincidental, but the categorical constructions are abstractions of (not only) these concepts.

Definition 13. *Let* $\mathbf{C}$ *be a category and* $A, B \in \mathcal{O}_{\mathbf{C}}$ *be objects. The* ***product*** *of two objects* A *and* B *consists of an object* P *and morphisms* $\pi_1 : P \to A$, $\pi_2 : P \to B$ *such that for any object* $W \in \mathcal{O}_{\mathbf{C}}$ *with morphisms* $f : W \to A$, $g : W \to B$, $f, g \in \mathcal{M}_{\mathbf{C}}$. *Moreover, there exists a unique morphism* $u : W \to P$, $u \in \mathcal{M}_{\mathbf{C}}$ *such that the diagram commutes, that is, such that* $f = \pi_1 \circ u$ *and* $g = \pi_2 \circ u$.



The object P *is usually denoted by* $A \times B$ *and the morphisms* π_1 *and* π_2 *are referred to as* ***projections****.*

Definition 14. *Let* $\mathbf{C}$ *be a category and* $A, B \in \mathcal{O}_{\mathbf{C}}$ *be objects. The* ***coproduct*** *of two objects* A *a* B *consists of an object* Q *and morphisms* $i_1 : A \to Q$, $i_2 : B \to Q$ *such that for any object* $Z \in \mathcal{O}_{\mathbf{C}}$ *with morphisms* $f : A \to Z$, $g : A \to Z$, $f, g \in \mathcal{M}_{\mathbf{C}}$. *Moreover, there exists a unique morphism* $u : Q \to Z$, $u \in \mathcal{M}_{\mathbf{C}}$ *such that the diagram commutes, that is, such that* $f = u \circ i_1$ *and* $g = u \circ i_2$.



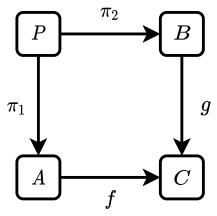
The object Q *is usually denoted by* $A + B$ *and the morphisms* i_1 *and* i_2 *are referred to as* ***injections****, even though they do not need to be injective.*

Remark: In the literature, coproduct is also referred to as a sum [124].

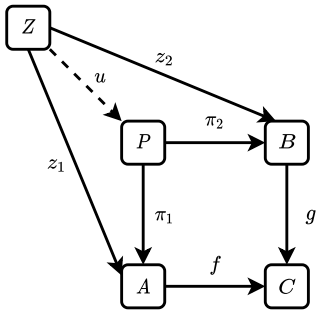
Definition 15. *A morphism* $f : A \to Z$ *is* ***complementable*** *if and only if there exists a morphism* $g : B \to Z$ *such that* $Z \cong A + B$, *where* f *and* g *are injection morphisms. We refer to the morphism* g *as the* ***complement*** *of* f *and the object* B *is often denoted as* $Z - A$.

Definition 16. *Let* $\mathbf{C}$ *be a category, let* $A, B, C \in \mathcal{O}_{\mathbf{C}}$ *be objects, and let* $f : A \to C$ *and* $g : B \to C$, $f, g \in \mathcal{M}_{\mathbf{C}}$ *be morphisms.*

The ***pullback*** *of* $A \xrightarrow{f} C \xleftarrow{g} B$ *is* $A \xleftarrow{\pi_1} P \xrightarrow{\pi_2} B$ *such that* $f \circ \pi_1 = g \circ \pi_2$, *i.e., such that the square commutes.*



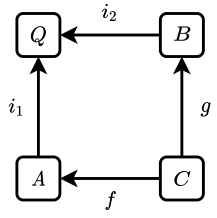
Moreover, it must hold that for any object $Z \in \mathcal{O}_{\mathbf{C}}$ and two morphisms $z_1 : Z \to A$ and $z_2 : Z \to B$, $z_1, z_2 \in \mathcal{M}_{\mathbf{C}}$ such that $f \circ z_1 = g \circ z_2$, there exists a unique morphism $u : Z \to P$ such that $z_1 = \pi_1 \circ u$ and $z_2 = \pi_2 \circ u$.



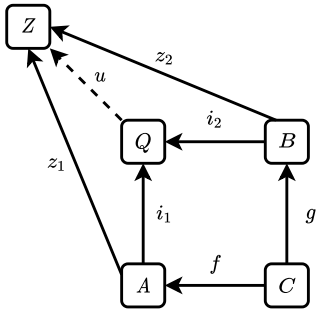
The object P is usually denoted by $A \times_C B$.

Definition 17. *Let $\mathbf{C}$ be a category, let $A, B, C \in \mathcal{O}_{\mathbf{C}}$ be objects, and let $f : C \to A$, $g : C \to B$, $f, g \in \mathcal{M}_{\mathbf{C}}$ be morphisms.*

*The **pushout** of $A \xleftarrow{f} C \xrightarrow{g} B$ is $A \xrightarrow{i_1} Q \xleftarrow{i_2} B$ such that $i_1 \circ f = i_2 \circ g$, that is, such that the square commutes.*



Moreover, it must hold that for any object $Z \in \mathcal{O}_{\mathbf{C}}$ and two morphisms $z_1 : A \to Z$ and $z_2 : A \to Z$, $z_1, z_2 \in \mathcal{M}_{\mathbf{C}}$ such that $z_1 \circ f = z_2 \circ g$, there exists a unique morphism $u : Q \to Z$ such that $z_1 = u \circ i_1$ and $z_2 = u \circ i_2$.



The object Q is usually denoted by $A +_C B$.

Note that universal constructs are characterised by the existence of a unique morphism u. This feature is referred to as the universal property UP or the universal mapping property UMP. Finally, as universal constructs are given by a universal property, they are unique up to the unique isomorphism [123].

Example A.9. Let $C = \{a, b, c\}$ and $R = \{1, 2\}$ be sets. Figure A.8 illustrates an example of the product of sets C and R, i.e., the Cartesian product $C \times R = \{(c, r) | c \in C, r \in R\}$ together with the projections $\pi_1 : C \times R \to C$ (red morphism) a $\pi_2 : C \times R \to R$ (green morphism). Moreover, if there is another candidate product (not necessarily a Cartesian product), e.g., $W = \{k\}$ with projections $f : W \to R$ (yellow morphism) and $g : W \to C$ (blue morphism), then there exists a universal mapping $h : W \to R \times C$ (purple morphism) such that the diagram commutes.

At first sight, the universal property of products may not be intuitive. As an example, let us give a real-world meaning to the sets C, R and W. Imagine that we have a set of game pieces W that we want to place on a board with assigned coordinates $Column = C$ and $Row = R$, i.e., each field can be identified as a pair $(c, r) \in Column \times Row$, where $\pi_1 : Column \times Row \to Column$ returns the column coordinate and $\pi_2 : Column \times Row \to Row$ returns the row coordinate. Placing the game pieces at the coordinates $c \in Column$ and $1 \in Row$, i.e., an application of the functions $f : W \to Column$ and $g : W \to Row$ corresponds to applying function $h : W \to Column \times Row$, i.e., selecting the board field $(c1)$.

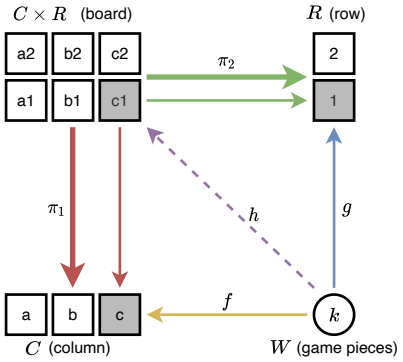


Figure A.8: An example of a product corresponding to Cartesian product of sets

Another example of a product of sets C and R is the Cartesian product $R \times C = \{(r, c) | r \in R, c \in C\}$. It is easy to prove that due to the universal product property, both products are isomorphic, i.e., $R \times C \cong C \times R$.

Finally, there also exists products of more than just two sets (i.e., objects). For example, let A, B, and C be sets. Then, e.g., $A \times B \times C = \{(a, b, c) | a \in A, b \in B, c \in C\}$ is the product of these sets. □

Example A.10. Let $B = \{1, 2, 3, 4\}$ and $W = \{1, 2\}$ be sets. Figure A.9 illustrates an example of the coproduct of sets B and W, i.e., disjunctive union $B + W = (B \times \{\blacksquare\} \cup (W \times \{\square\})$, where $\{\blacksquare, \square\}$ denotes the origin of the element (i.e., $\blacksquare$ is assigned to each $b \in B$ and $\square$ is assigned to each $w \in W$), together with the inclusions $i_1 : B + W \to B$ (red arrows) a $i_2 : B + W \to W$ (green arrows). Moreover, if there is another candidate coproduct (not necessarily a disjunctive union), e.g., $Z = \{r, n, b, k, q\}$ with inclusions $f : B \to Z$ (yellow arrows) and

$g : W \to Z$ (blue arrows), then it holds that there exits a universal mapping $h : B + W \to Z$ (purple arrows) such that the diagram commutes.

Let us again illustrate the coproduct using a real example. Suppose the set B represents the (sub)set of black chess pieces and the set W represents the (sub)set of white chess pieces. The set of all pieces is their disjunctive union $B + W$, i.e., the (sub)set of chess pieces. A candidate coproduct may be the set $Z = \{rock, knight, bishop, king, queen\}$, together with morphisms $f : A \to Z$ and $g : B \to Z$, which determine the type of the pieces, i.e., r is a *rock*, n is a *knight*, b is a *bishop*, k is a *king*, and q is a *queen*. Then there exists a unique morphism $h : B+W \to Z$ which states that the type of a piece can be determined for all chess pieces, i.e., the diagram commutes.

Moreover, similar to Example A.9, there can be multiple coproducts that are isomorphic to each other. It is also possible to construct a coproduct for more than two sets (i.e., objects). □

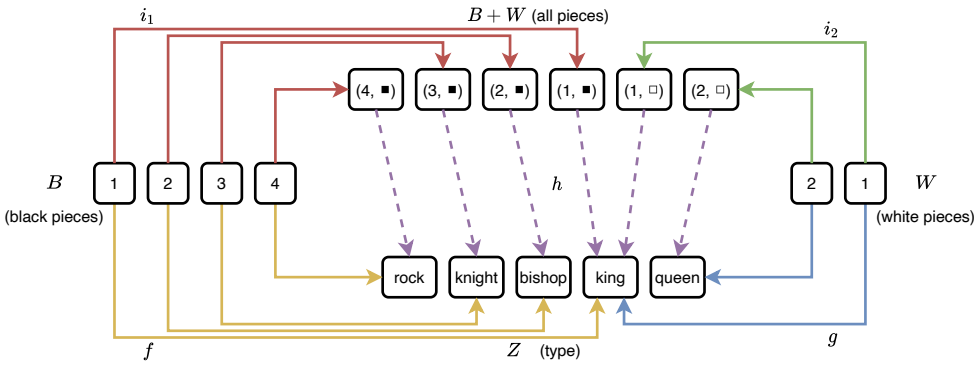


Figure A.9: An example of a coproduct corresponding to disjoint union of sets

Example A.11. Let $A = \{1, 2, 3, 4\}$ and $B = \{3, 4, 5, 6\}$ be sets and let $C = \{1, 2, 3, 4, 5, 6\}$ be their union, i.e., there exist inclusive mappings $f : A \to C$ and $g : B \to C$.

The pullback of $A \xleftarrow{f} C \xrightarrow{g} B$ is, e.g., $A \xrightarrow{\pi_1} P \xleftarrow{\pi_2} B$, where P consists of elements p such that $f(a) = g(b) = p$, i.e., $P = \{3, 4\} = A \cap B$, and π_1, π_2 are the projections (see Figure A.10 (a)). It is easy to verify that the diagram A.10 (b) commutes.

Moreover, if there exists another pullback candidate, e.g., $A \xrightarrow{z_1} Z \xleftarrow{z_2} B$, where, e.g., $Z = \{3\}$ such that $f(3) = g(3)$, then there exists a universal (injective) mapping $h : Z \to A \cap B$ such that the diagram commutes. □

Example A.12. Let $A = \{1, 2, 3, 4\}$ and $B = \{3, 4, 5, 6\}$ be sets and let $C = \{1, 2, 3, 4, 5, 6\}$ be their intersection, i.e., there exist projections $f : C \to A$ and $g : C \to B$.

The pushout of $A \xrightarrow{f} C \xleftarrow{g} B$ is, e.g., $A \xleftarrow{i_1} Q \xrightarrow{i_2} B$, where Q consists of elements q such that $q \in A + B$, $A + B$ being a disjoint union of A and B and $a \in A$ are identical to $b \in B$ if there exists $c \in C$ such that $f(c) = a$ and $g(c) = b$, hence we obtain a union $A \cup B$ (see Figure A.11 (a)). It is easy to verify that the diagram in Figure A.11 (b) commutes. □

Finally, Table A.4 summarises the application of natural transformation in approaches representing data at the conceptual or logical level and in approaches performing data migration.

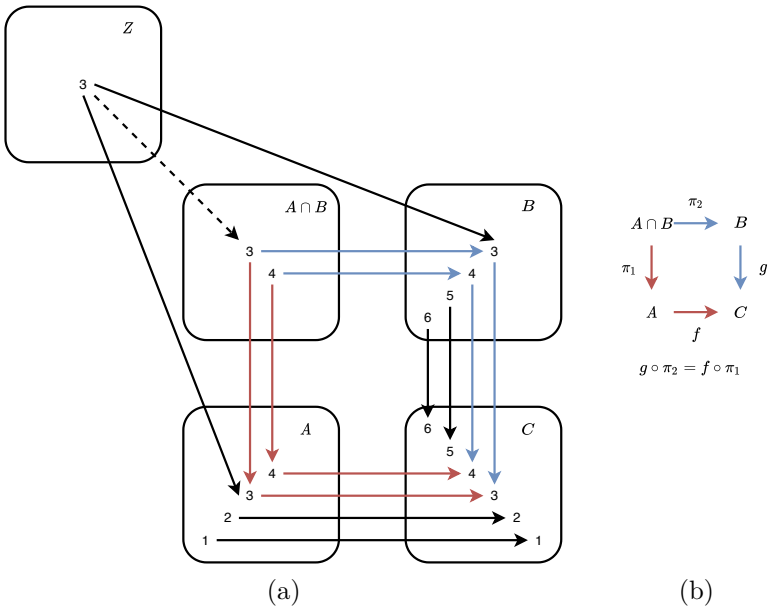


Figure A.10: An example of a pullback corresponding to an intersection of sets

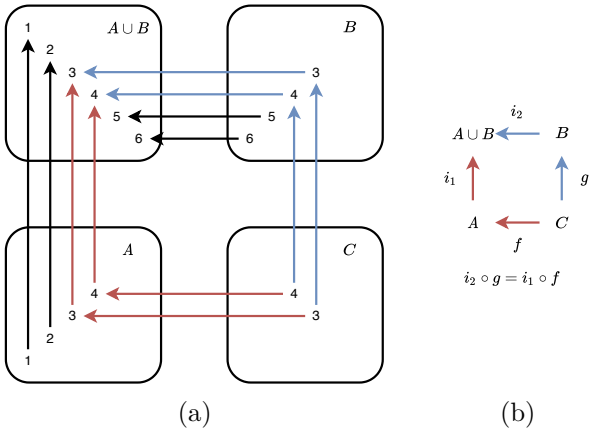


Figure A.11: An example of a pushout corresponding to union of sets

Table A.4: Application of universal constructions

	Lippe and Ter Hofstede [18]	**CGOOD** [19]	**Spivak et al.** [20]	**APG** [21]	**MM-(evo)cat** [3, 5]
Product (Definition 13)	Complex identifier; Set	-	Querying (not discussed)	Querying (not discussed)	Complex identifier
Coproduct (Definition 14)	-	-	Querying (not discussed)	Querying (not discussed)	Multiple identifiers
Complemetable morphism (Definition 15)	Inheritance	-	-	-	-
Pullback (Definition 16)	-	-	Querying (not discussed)	Querying (not discussed)	Joining of data
Pushout (Definition 17)	Generalisation	Addition of data	Querying (not discussed)	Querying (not discussed)	Addition of data