*FRAME INFORM SYSTEMS*

LDBD Research Report ◇ May 1997

# Generalized sketches as an algebraic graph-based framework for semantic modeling and database design

*Zinovy Diskin*

Laboratory for Database Design
FIS/LDBD-97-03, May 1997

Elizabetes 23, Riga, LV1234, Latvia ◇ Phone: (371 7)320323 ◇ Fax: (371 7)320376 ◇ diskin@fis.lv

# Generalized sketches as an algebraic graph-based framework for semantic modeling and database design

Zinovy Diskin *

Lab. for Database Design
*Frame Inform Systems*
Elizabetes Str. 23, Riga
LV-1234, Latvia
*E-mail:* diskin@frame.riga.lv
*Fax:* (371 7)828036

**Abstract.** A graph-based specification language and the corresponding machinery are described as stating a basic framework for semantic modeling and database design. It is shown that a few challenging theoretical questions in the area, and some of hot practical problems as well, can be successfully approached in the framework. The machinery has its origin in the classical sketches invented by Ehresmann and is close to their generalization recently proposed by Makkai.
There are two essential distinctions from Makkai's sketches. One consists in a different – more direct – formalization of sketches that categorists (and database designers) usually draw. The second distinction is more fundamental and consists in introducing *operational* sketches specifying complex diagram operations over ordinary (predicate) sketches, correspondingly, models of operational sketches are *diagram algebras*. Together with the notion of parsing operational sketches, this is the main mathematical contribution of the paper. It is demonstrated that it gives rise to a proper design of graphical query languages. On the other hand, the machinery presents an immediate formalization of diagram chasing – the major tool of categorical reasoning and, hopefully, of the nearest future semantic modeling techniques.

* Supported partially by Grants 93.315 and 93.254 from the Science Council of Latvia and partially by Grant 314/94-72II (the scheme for research collaboration with Eastern Europe) from the Chalmers University (Sweden)

## Table of Contents


**1 Introduction and motivating discussion** . . . 1
- 1.1 Why category theory? . . . 1
- 1.2 Why generalized sketches? . . . 2
- 1.3 Why operational sketches? . . . 5

**2 The world consists of sets and functions: Data definition via diagram predicates** . . . 6
- 2.1 Semantic modeling via sketches . . . 6
  - Internal structure of objects via arrow diagrams . . . 6
  - Sketches vs ER-diagrams: an example . . . 7
  - Discussion . . . 8
- 2.2 Formalizing sketch syntax . . . 10
- 2.3 Formal semantics for sketches . . . 14

**3 Databases as graphical algebras: Data derivation via diagram operations** . . . 15
- 3.1 Operational sketches in database design . . . 15
  - Examples . . . 15
  - Discussion . . . 16
- 3.2 Formalizing operational sketches . . . 17
  - Preliminary description . . . 17
  - Signatures and algebras . . . 18
  - Semantics . . . 19
  - Query mechanism . . . 20
  - Operational sketches and their parsing . . . 20

**4 Towards graph-based finite model theory: Sketching metafinite structures** . . . 23

**5 Conclusion** . . . 25


## List of Tables


1 A collection of diagram predicates . . . 32
2 A collection of diagram operations . . . 33


## List of Figures


1 Primitive ER-diagrams and generalized sketches . . . 3
2 From Ehresmann's sketches to generalized sketches . . . 3
3 Several simple sketches . . . 7
4 Sketches vs ER-diagrams . . . 28
5 Extraction of indexed sketch from a visual sketch . . . 29
6 Sketches with operation markers . . . 30
7 Example of schema integration . . . 30
8 Sketch parsing . . . 31

# 1 Introduction and motivating discussion

The core of a complex information system is a database (DB) or a net of DBs. In the modern view, DB presents an integral model of the real world fragment (universe) upon which the information system is to be built. Hence, a crucial initial step for the proper information system design is to specify the universe in abstract and formalizable terms suitable for further transformation into lower-level software specifications. The problem itself and the corresponding software engineering (SE) discipline pursuing it are called *semantic modeling* (SM), or else *conceptual* or *information modeling*.

## 1.1 Why category theory?

**1.1.1 Problem.** A semantic specification has to compress an informal description of non-toy universe into a compact comprehensible text suitable for communication between the database designer, experts on the universe and users of the information system to be designed. A natural (and practically without alternatives) choice is to use graphical languages, and indeed, a vast variety of graphical notational systems were developed in SM. In particular, extremely popular are *entity-relationship diagrams* (see, *eg*, [4]) which became a kind of *de facto* standard in the area.

On the other hand, a semantic schema should be formalized in order to be easily (the more so, computer-aidedly) transformed into software specifications on the further stages of design. Thus, a good semantic language should be *graph-based, formalizable* and sufficiently *expressive* to capture all the peculiarities of the real world. To the current state of the art in SM, the synthesis was not achieved: specification languages employed are either semi-formal (or, worse, quasi-formal) or have a very restricted expressive power, or both, so that today a database designer is forced to alter between easy-to-use but scarcely formalized graph-based approaches and logically clear and perfect but inflexible low-level relational languages. Due to the evident tendency of any practitioner towards the former, what exists out in the field today is "cookbook approaches ... poorly formed and having too many subjective human considerations" (Navathe, [32]). In addition, the lack of formal semantics converts semantic schemas into graphical interfaces to low-level relational schemas rather than high-level semantic specifications intended in the very setting of SM goals.

Furthermore, a direct consequence of restricted expressive power of modern semantic languages is that a great part of data semantics is programed immediately without specifying in the semantic schema. This gives rise to the infamous phenomenon usually phrased as that *semantics of data is hidden in the application code* (see, *eg*, [21]), which has induced a whole field of research and practical activity – the so called reverse engineering. $\diamondsuit$

**1.1.2 Solution?** Development of languages integrating the desired properties above is just in the focus of category theory (CT). Its experience has shown that dealing with graphical yet formalizable specifications requires a special kind of thinking (*arrow thinking*!) and the corresponding machinery, and the success hardly can be achieved with *ad hoc* approaches built from scratch. This explains the far from ideal situation in the modern SM and, on the other hand, suggests that CT should possess a great potential for software engineering applications. Indeed, it was demonstrated in [13, 29, 8, 10, 19] that even initial categorical arrangement of standard specification languages used in SM leads to *practically useful results*.

This success is remarkable and supports the claim. It would be however too optimistic to hope that CT in its current state is immediately applicable for systematic using in software engineering: actually it needs development of some core issues to make adaptation of the arrow framework for practical needs real. $\diamondsuit$

### 1.2 Why generalized sketches?

**1.2.1** Categorists prefer to work with closed structures rather than their generators: categories instead of graphs, monads instead of terms and equations, classifying categories instead of formulas and axioms and so on. No doubts, availability of presentation-free descriptions is a source of power of CT in metamathematical studies, and also can be extremely useful in specifying architecture of information systems ([20]), that is, in metadata modeling. As for data modeling, it requires *finite, effective* and *comprehensible* presentations of complete structures, and the corresponding specification machinery. All characteristics above are essential and even crucial for applications: this is evident for the former two but comprehensibility is also a must if specifications are intended for real usage by software engineers. $\diamondsuit$

**1.2.2** A presentation-oriented trend in category theory is associated with the concept of *sketch* invented by Ehresmann. Ehresmann's sketches were directly intended for effective presentation of complex structures but unfortunately they cannot be utilized in SE in a immediate way.

Let us consider a simple example. The notion of relation is a major one for the modern database technology, and many techniques of database design are based on various dialects of entity-relationship diagrams. The standard view on relations is to consider a relation $R$ between entity sets $E_1, \ldots, E_m$ as a set of tuples (relationships) over $E_1, \ldots, E_m$, that is, as a relationship set. Correspondingly, in semantic schemas relationship nodes are specially designated, say, by a diamond in contrast to rectangle entity nodes: a sample ER-diagram is shown on Fig. 1(a). This notation turned out to be very successful so that now DB designers cannot think their activity without ER-diagrams, and have developed a plenty of notational systems extending them in various directions. However, the very specification principle when a relation is a set of tuples, a grouping set (powerset) is a set of subsets *etc*, leads to serious problems due to the so called *semantic relativism*. The point is that another user can view the same objects differently, for example, a lawyer would probably consider *Married*-objects as entities by themselves and view the universe as presented by the ER-diagram on Fig. 1(b). However, a corner-stone of the DB technology is that different users are served by the same database, and thus local views must be integrated into a global semantic schema (moreover, a proper integration is often is a key to the correct DB design). The question is how one should consider the *Married*-node in the global schema. An abundance of papers were devoted to the integration problem but a sufficiently general consistent approach beyond *ad hoc* solutions was not found (see discussion in [10]).

Well, CT gives a clue: a relation is a set $R$ together with a jointly monic family of arrows (projections) $p_i\colon R \to E_i$, $i = 1, \ldots, m$, so that specifying internal structure of $R$-objects is removed to an arrow diagram adjoint to the node, and the problem of entity-relationship clash disappears.[2] A standard

[2] Actually, this is a very rough description. It was shown in [18] that the common intuition of distinguishing between entities and relationships has two aspects: structural, which was just described, and evolutional, which can be captured in the framework of variable sets semantics.

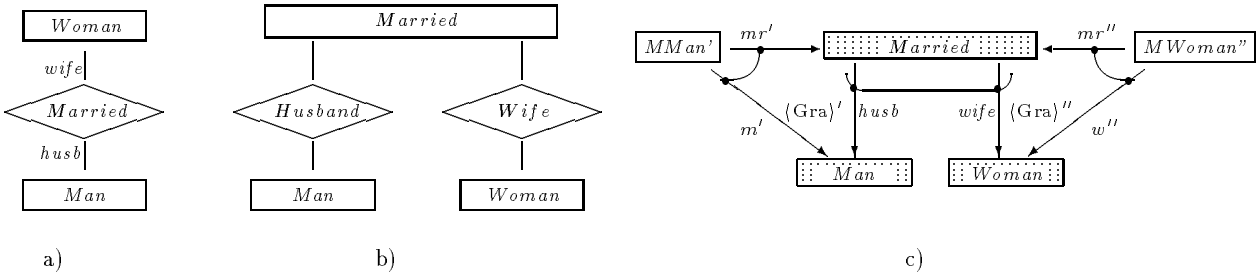



**Fig. 1.** Primitive ER-diagrams and generalized sketches

categorical way of specifying jointly monic families is to work in a finitely complete category and draw a finite limit sketch shown on diagram Fig. 2(a.1) below: arcs denote limit cones so that $E$ is a Cartesian product and the square $RRRE$ is pull-back, that is, $r$ is monic via standard arguments. This description is mathematically elegant but from the view point of applications it has two drawbacks:

(i) the specification involves the node $E = E_1 \times \ldots \times E_m$ whose extension is (fortunately!) not required to be stored in the database;

(ii) a simple property of the function $r$ is specified in a very indirect way with drawing new auxiliary nodes and arrows.

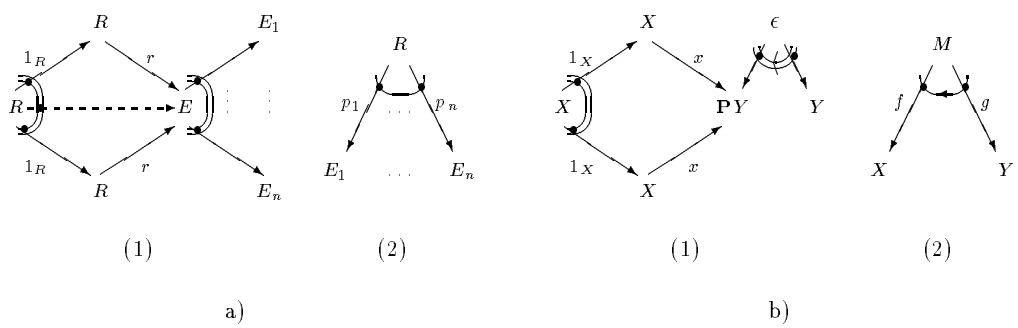



**Fig. 2.** From Ehresmann's sketches to generalized sketches

A much more direct way of specifying relations is to introduce a predicate of being jointly monic for families of arrows with a common source, and then declare that the family $(p_1, \ldots, p_m)$ satisfies the predicate. Such a declaration is visualized by labeling the corresponding diagram with a marker, say, an arc, and then the specification takes the simple form shown on diagram (a.2). $\diamondsuit$

**1.2.3** A similar problem occurs when one needs to specify a set $X$ consisting of subsets of $Y$ (the so called *grouping* construct in semantic modeling terms). A standard categorical approach is to specify first the powerset of $Y$, $\mathbf{P}Y$, by universal properties and then specify a monic arrow: $X \to \mathbf{P}Y$ (Fig. 2(b.1)). However, appearance of the full powerset node in a database specification leads to intractability and may frighten any database designer. In contrast, a direct tractable way of specifying $X$ as subpowerset of $Y$

is presented on Fig. 2(b.2) where the directed arc denotes the following property of the pair $(f, g)$:

$$\text{for any } x, x' \in X, \text{ if } \{g(m) \mid m \in f^{-1}(x)\} = \{g(m) \mid m \in f^{-1}(x')\} \text{ then } x = x' \ ,$$

so that $X$ is embedded into $\mathbf{P}Y$. $\diamondsuit$

**1.2.4** The examples above suggest to introduce a general notion of diagram predicate: syntactically, it is a marker having an *arity shape* designating those graphs on which the marker can be hung, semantically, a marker is interpreted as a certain property of systems of sets and functions whose configuration fits in with the shape. A data definition language is then set by a collection (signature) of diagram predicates and a specification appears as a graph in which some diagrams are labeled by markers from the signature. Such specifications can be considered as an immediate generalization of Ehresmann's sketches and were proposed by Makkai (see [31] for references) and independently by the present author in [7]. Makkai called them *generalized* sketches, we will also use the name *$\Pi$-sketch* with $\Pi$ being the name of the signature. Furtheron in the paper, the term *sketch* means generalized sketch while classical sketches familiar to the CT-community will be called Ehresmann's sketches (this meets Bourbaki's precepts on terminology). $\diamondsuit$

**1.2.5** Marked diagrams in a sketch must be easily recognizable, otherwise sketches lose their benefits of evident graphical specifications. So, the shape graph $D$ of a predicate (marker) should occur in the carrying graph $G$ of the sketch without topological deformations, that is, as is standard in CT, a diagram appears as a visually clear presentation of a graph morphism $\delta\colon D \to G$. For example, on Fig. 2(a) the arrow square $RRRE$ denotes a graph morphism whose domain is the square shape of the pull-back predicate. Note, however, that the two left arrows of the square $RRRE$ are also marked: $1_R$ is an arrow property rather than name. What is the shape of (the marker of) this property?

Usually it is taken to be the loop graph but then any arrow declared as identity must be a loop and the simple sketch Fig. 2(a) becomes syntactically incorrect. Probably, for a particular predicate the problem can be avoided with some trick but a proper general solution requires to modify the very concept of sketch: shape graphs of predicates and carrier graphs of sketches must be labeled by names and diagrams are graph morphisms compatible with naming. For example, the shape of identity is an arrow between two different nodes with the same name so that an identity diagram in a labeled graph is an arrow for which labels (names) of the source and target nodes coincide.

This modification leads to three mutually related notions of (generalized) sketches: *visual sketches* which categorists (and now database designers in our laboratory as well) usually draw on a paper (computer display), *indexed sketches* extracted from visual sketches by forgetting inessential details of visual presentation of diagrams and, finally, *ordinary sketches* which are pure logical constructs hidden in visual sketches (details will be given in section 2.2). Just ordinary sketches are semantically valid (and present an immediate generalization of Ehresmann's sketches) whereas visual sketches are a kind of human-oriented interface to ordinary sketches. What are indexed sketches for? There are two reasons. The first goes from the computer: indexed sketches are an immediate output of abstracting visual sketches, and their data structure is easier for computer processing. The second goes from mathematics: categories of indexed sketches behave themselves better than that of ordinary sketches, it was shown by Makkai in [31]. It is worth noting that in Makkai's framework there are no visual sketches so that appearance of indexed

sketches looks somewhat mysterious and is justified by only internal mathematical reasons. Thus, pure categorical reasons, on one hand, and practice of software engineering, on the other, have led to the same formal construct – a remarkable coordination. ♢

### 1.3 Why operational sketches?

As soon as some specification language is proposed, it should be endowed with an appropriate deduction machinery, that is, with a mechanism of augmenting any given specification with additional items denoting derived information. In the DB terminology, this is the well known issue of query languages and querying mechanism. What is less recognized in the DB literature (but was at once noticed in [29]) is that derived items are often necessary for specifying constraints on data and so, in general, operations (queries) may be involved in data definition schemas as such. Moreover, for complex universes whose schema appears as the result of integrating local view schemas, the presence of queries in data schemas is a rule rather than exception ([8, 10]). In this context, the discussion below is motivated by the problem of specifying queries and query languages rather than their computational complexity and expressive power.

**1.3.1** The relevance of algebraic framework for DB theory is well known and goes back to the classical Codd's papers (see [34] for a more advanced treatment). However, SM needs *graph-based* algebraic machinery, and a natural framework for developing the latter is the framework of categorial logic and algebra. In this framework, given a logical doctrine $\mathcal{D}$, any specification (semantic schema) $S$ can be completed up to its classifying category – a (usually infinite, monstrous) schema $\mathbf{der}S \in \mathcal{D}$, and queries against $S$ become finite subgraphs of $\mathbf{der}S$ ([29, 17]). It was proposed in [29] to use internal logic of $\mathcal{D}$ as a query language. Statements of the internal logic can indeed serve as queries but in this way of specifying queries their algebraic nature is hidden. In addition, quite simple queries may turn out to be too bulky in their string-based internal logic formulation and heavy to use in communication between DB designer and DB users, this contradicts the very setting of SM as an engineering discipline.

Semantic schemas are graphical images, and so a natural query language is to use diagram operations definable in $\mathcal{D}$ in the role of queries. However, though some initial ideas on diagram operations and algebras were developed in CT ([6, 22]), a consistent general framework is not built (in a sharp contrast with the well developed presentation-free machinery of monads (see, *eg*, a survey [35]). ♢

**1.3.2** The issue becomes especially actual in the task of view integration. A major difficulty in it is to resolve the so called *structural conflicts* between views when different DB users model the same information in different ways (see, for example, Fig. 1(a,b) again). A diversity of such conflicts was described in the literature, and a large body of research was devoted to their managing (see, *eg*, [39, 38] for a survey) but they still remain a stumbling block for the majority of view integration methodologies. It was however shown in [8, 10] that structural conflicts between views are nothing but a well known algebraic phenomenon when some basic items of one theory are derived items in another theory. (For example, in situation of Fig. 1, relations *Wife*,*Husb* of the right view are nothing but graphs of functions *wife*, *husb* of the left view as is shown on Fig. 1c). To integrate schemas one augments them with derived items and declares the corresponding set of equations between basic/derived items occurring in different local schemas (*eg*, RightView.*Husband*= **Graph**(LeftView.*husb*) where **Graph** denotes a simple graphical

operation of building the graph of function). The resulting specification is then appears as a set of defining relations over some set of generators (items of local schemas). So, the infamous problem of view integration is reduced to a well known way of specifying algebras by generators and defining relations and can be effectively managed in some algebraic framework([8, 10]). However, in contrast to the ordinary universal algebra where generating collections are sets, and operations are applied to lists of their elements, here we deal with generating graphs and operations are applied to diagrams in these graphs. $\diamondsuit$

Thus, what SM needs is a well stated machinery of graphical operations and graphical algebras. In the sketch framework we advocate, they take the form of diagram operations over sketches and diagram algebras. Principal questions are: what is an arity of diagram operation, how to specify compositions of such operations, what is an algebra in a given signature of diagram operations and, crucially for applications, how to set such algebras in a effective way. It will be shown in section 3 that effective presentations of diagram algebras can be described graphically in a comprehensible way by a special kind of sketches called *operational sketches*.

The rest of the paper is an interplay between (i) *sketching* toy but characteristic examples of semantic modeling, (ii) their extrapolation to general discussions and (iii) some details of precise formalization. With respect to mathematics, the focus is on definitions rather then theorems. Some open problems of finding an appropriate definition are formulated.

## 2 The world consists of sets and functions: Data definition via diagram predicates

Even an overall glance at graphical images currently used in semantic modeling shows an abundance of various kinds of conventional graphical constructs, symbols, labels and markers. Every specialist in semantic modeling and every DB designer uses that kind of semantic schemas which one finds more suitable and convenient for one's purposes. This is natural and reasonable but the diversity of notational systems makes comparing and relating semantic specifications designed in different models an extremely difficult task and, in particular, results in the infamous problem of heterogeneous schema integration. Nowadays the issue becomes especially hot due to the rapid advance of cooperative information systems.

An abundance of *ad hoc* approaches to semantic modeling leads to serious problems in software design and engineering. In contrast, many of specificational problems can be managed in a coherent way if one treats semantic specifications as (generalized) sketches, *ie*, theories in a graph-based logic of diagram predicates and operations.

### 2.1 Semantic modeling via sketches

#### Internal structure of objects via arrow diagrams

**2.1.1** The chief idea of the sketch approach to semantic modeling is to consider classes of real world objects homogeneous sets while all the information about internal structure of objects is moved into arrow diagrams adjoint to classes. This approach constitutes the essence of the *arrow thinking* underlying CT.

Two important examples of arrow treatment of conventional SM constructs were described in introduction, some others are presented in Table 1. Then, for example, the following five pictures (sketches) on Fig. 3 specify the node $X$ as (a) relation, (b) disjoint sum, (c) union, (d) the image of a given function and (e) subpowerset. In more detail, any extension mapping $e$ which assigns sets and functions to the sketch items and satisfies the conditions expressed by the markers will necessarily provide the corresponding structure of the set $X$ and its elements. $\diamondsuit$

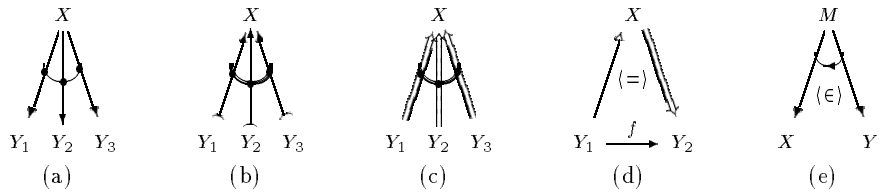



**Fig. 3.** Several simple sketches

**Sketches vs ER-diagrams: an example**

We consider a simple example demonstrating the sketch treatment of ER-diagrams, *ie*, specifying basic constructs of ER-model: relationships, weak entity sets and attributes via diagram predicates.

**2.1.2** The sample semantic situation we wish to model is as follows. Suppose, the user is interested in information about men, married couples and married women of some town for, say, the last 50 years. A rough view on the universe is described by the ER-diagram on Fig. 4 in a conventional notation (see, *eg*, [12]). Semantics of nodes and attributes is hopefully clear from its names, *MDate,DDate* are dates of marriage and divorce (the latter is optional). *BDate* is a tuple type determined by its own attributes $Day$:`Int`, $Month$:`Int`, $Year$:`Int`. The domain of the attribute *Character* is a set consisting of three values $a,c,q$. The double frame decorating the class *Woman* denotes that it is of the so called *weak entity type*: users are assumed to be interested not in all the women (in the universe) but only those which are or have been married.

Note, the ER-diagram does not reflect the reality exactly: the diamond means that *Married*-objects are identified by pairs $(wife, husb) \in Woman \times Man$ whereas, in general, this is not the case. Indeed, it is well possible that a married couple got divorced and then married again. Thus, the correct identifier is the triple (*wife,husb,mdate*). The diagram is drawn as shown since in the notational ER-standard there is no graphical means to express the required semantics. In contrast, this can be easily done with the sketch machinery: the sketch specifying the situation is depicted on the lower figure (see Table 1 for the meaning of the marked diagrams). Note the monic marker on the key attribute *ssn* and the cover marker on the arrow *wife*.

Comparing the ER-diagram and the sketch, one can observe the unfortunate ER-tradition to visualize diagram predicates by labeling nodes instead of arrows the diagram consists of. One example is visualization of separating family of arrows by framing the source node, another (sharp) one is visualization of cover arrows by framing their target nodes. Unfortunately, in a more or less extent such a practice is

common for many notational systems used in semantic modeling. It makes discovering formal semantics of many conventional models an extremely difficult task and blocks their efficient using in CASE-tools. ♢

**2.1.3** Rectangle nodes are *abstract classes* whose extensions should be stored in the DB: this is additionally pointed by small dots filling-in rectangles. Oval nodes are predefined *value domains* whose semantics is *a priori* known to the DBMS. For the sketch approach, $\langle$`Int`$\rangle$ and $\langle\{$`a,c,q`$\}\rangle$ are *markers* (in our precise sense) hung on the corresponding nodes, that is, constraints imposed on their intended semantic interpretations. For example, if a node is marked by $\langle$`Int`$\rangle$ its intended semantics is the predefined set of integers. So, abstract class nodes are labeled by names, and value domain nodes (datatypes) are labeled by markers expressing their intended semantics (while their names become abundant and can be omitted).

Note also that extensions of all value domains besides $\langle\{$`a,c,q`$\}\rangle$ are computable rather than storable and their semantics can be explained in the abstract data type (ADT) framework (see, *eg*, [23]). In fact, $\langle$`Int`$\rangle$, $\langle$`Date`$\rangle$ *etc* are names of (quasi)equational specifications whose initial semantics gives the intended extension. One can use sketches for specifying ADT too ([41]), then a value domain node is an entry to some complex sketch. As for $\langle\{$`a,c,q`$\}\rangle$-domain, its extension has to be stored but together with the database schema rather than with the database extension. ♢

**2.1.4** Some words about optional attributes would be useful. A proper approach is to treat them as partial functions. However, as soon as partial functions appear the semantic framework changes. Indeed, partiality of a function is not an arrow property when arrows are interpreted as total functions. Quite the reverse, totality is a property a partial function may have. Thus, to handle partial functions in a systematic way one must interpret all arrows as partial functions by default and introduce a special arrow predicate of being total. In other words, the intended semantic interpretation is switched to the topos of sets and partial functions. Similar considerations apply for multi-valued functions and so one comes to interpreting sketches in the topos of binary relations between sets. Thus, it appears that the most appropriate categorical setting for semantic modeling is to model data by sketches interpretable in Peter Freyd's allegories ([24]).

In this paper we do not wish to cope with a general setting described above. So, in the example above we choose a naive way of treating the optional attribute *ddate* by lifted set $\texttt{Date}^{-}$ as shown on Fig. 4. ♢

### Discussion

**2.1.5** The sketch specifications pattern is strict and compact: sketches are graphical constructs consisting of three kinds of items: (i) nodes, to be interpreted as sets, (ii) arrows, to be interpreted as functions between corresponding sets, and (iii) diagram markers, to be interpreted as constraints imposed on diagrams labeled by these markers. The experience of category theory shows that this idea provides an extremely powerful specification machinery. Moreover, through familiar topos arguments one can conclude that any (!) formalized data specification can be simulated by a sketch. ♢

**2.1.6** Sketches suggest the following way of managing heterogeneity of semantic models. Given a model $\mathcal{M}$, its vocabulary of specificational constructs is arranged to be come convertible into a signature of diagram predicates, $\Pi$, so that $\mathcal{M}$-specifications could be converted into $\Pi_{\mathcal{M}}$-sketches. The *diversity* of semantic models is thus transformed into a *variety* of sketch data models in different signatures. Indeed, sketches in different signatures are nevertheless sketches, and they can be uniformly compared and integrated via relating/integrating their signatures. Though the latter task is far from being trivial, it is precisely formulated and can be approached by mathematical (algebraic!) methods. Anyway, in many interesting particular cases the signature integration is easy.

The methodology of the sketch approach can be illustrated by the following analogy. In a sense, the place of sketches in the heterogeneous space of semantic models is comparable with that of the modern positional numeral systems in the general space of numeral systems including also zeroless positional systems (*eg*, Babylonian, Mayan) and a huge diversity of non-positional *ad hoc* systems (Egyptian, Ionian, Roman *etc*). The analogy we mean is presented in the table below.

| Specification paradigm | Data to be specified | Language | | Minimal language |
|---|---|---|---|---|
| | | Predefined base | Specification | |
| Sketches | Collections of sets and functions | Signature, $\Pi$ | $\Pi$-sketch | Ehresmann's sketches |
| Positional numeral systems | Finite cardinalities | Base of numeral system, $k$ | Positional $k$-number | Binary numbers |

$\diamond$

**2.1.7** The intention behind suggesting sketches is not to enforce all practitioners and researchers to use the same universal graphical language – let everyone use that collection of graphical constructs one likes. What is suggested concerns *what should be specified* in semantic schemas. After that, the question is how to visualize the specification constructs, that is, *what should be marked* in semantic schemas. The accurate distinction between logical specification and its syntactic-sugar visualization, and simultaneously their parallelism are the two key advantages of the sketch approach from the practical view point, they deserve some additional words.

Clear distinction between logical specification and its visualization is provided by the presence of formal semantics for sketches, and makes the sketch notation favorable in comparison with many of notational systems in semantic modeling. Of course, logic itself does not decide all of the problems: design of a good notational system is an art rather than technology. Nevertheless, the strict sketch pattern makes it possible to designate the formalizable part of the design task and state a precise basic framework for its consideration, *eg*, to formulate the *commutativity principle*: visualization of conjunction of diagram predicates should be conjunction of visualizations (*eg*, monic arrows are marked by decorating their tails whereas covers are decorated in their heads). This is a topic of future research.

Parallelism of specification and graphical visualization is provided by the graph-based nature of the sketch logic, and sharply distinguishes sketches from those semantic models which are externally graphical but internally are based on predicate-calculus-oriented string logics (a typical example is NIAM [27]). The repertoire of graphical constructs to be used in these models has to be bulk since every kind of logical formulas requires its special visualization. Configurations of these visualization constructs can be

rather arbitrary because there are no evident natural correlations between graphics and logical string-based formulas. In particular, this problem is actual for the modern CASE-tools whose underlying logic is relational, that is, is string-based.

Of course, there are situations when something can be easily described by a logical formula but hardly by a graphical image, and there are inverse situations as well. A good language has to combine graphical and string logics into a flexible notational mechanism. So, what we want to say about the opposition *graph-based – string-based* is that it is quite often in semantic modeling when arrow logic leads to much more comprehensible specifications than predicate calculus in its various string-based versions. ♢

### 2.2 Formalizing sketch syntax

Actually, the sketches we considered above are visual presentations of the corresponding logical constructs possessing a certain semantics. The latter matter in semantics and called *logical sketches*, the former matter in comprehending semantic specifications and called *visual sketches*. For computer processing both have to be formalized. So, when one draws a sketch (in some predefined signature $\Pi$ of diagram markers) on the computer display, one actually enters into the system some data structure called *visual $\Pi$-sketch*. Processing visual sketches for semantically valid tasks (*eg*, schema integration) needs to interpret them as *logical $\Pi$-sketches* which are "deeper" data structures free from accidental peculiarities of visual presentation.

There are two kinds of logical sketches – *indexed* sketches and *ordinary* sketches. Indexed sketches are an immediate output of abstracting visual sketches from inessential visualization details. Ordinary sketches is the result of a further abstraction and, in fact, is an immediate generalization of Ehresmann's sketches. The relations between these notions of sketch is described by the following chain of adjunctions

$$\mathbf{VSke}_{\Pi} \underset{\star}{\overset{j}{\underset{\longrightarrow}{\overset{\longleftarrow}{\top}}}} \mathbf{ISke}_{\Pi^\star} \underset{\bullet}{\overset{i}{\underset{\longrightarrow}{\overset{\longleftarrow}{\top}}}} \mathbf{OSke}_{\Pi^\star} \tag{1}$$

where $\Pi$ is some signature of (visual) diagram markers, $\Pi^\star$ is its logical counterpart and $\mathbf{VSke}_{\Pi}$, $\mathbf{ISke}_{\Pi^\star}$, $\mathbf{OSke}_{\Pi^\star}$ are categories of visual, indexed and ordinary $\Pi$-sketches; the upper/lower arrows are right/left adjoint functors, the upper ones are embeddings. These adjunctions will be explained below.

**2.2.1 Notation.** If $\mathbf{G}$ is a category and $X, Y$ are two sets of $\mathbf{G}$-objects, then $\mathbf{G}(X, Y)$ denotes the set $\{f\colon x \to y \mid x \in X, y \in Y\}$. Composition of morphisms is denoted by $f \blacktriangleright g$. Domain/codomain of a morphism $f$ are denoted by $dom(f)/cod(f)$ or $\Box f / f \Box$. ♢

**2.2.2 Visual graphs.** A *visual graph* is defined to be a finite graph $G_V$ whose items are labeled by *names* taken from some predefined pool in such a way that the labeling mapping can be considered as a covering graph morphism $G\colon G_V \to G_N$ into some graph of names, $G_N$. A *morphism* $h\colon G \to G'$ of visual graphs is a graph morphism from $G_V$ into $G'_V$ compatible with naming, in other words, $h$ is a pair $(h_V\colon G_V \to G'_V, h_N\colon G_N \to G'_N)$ s.t. $h_V \blacktriangleright G' = G \blacktriangleright h_N$ (the square commutes). Thus, the category of visual graphs, $\mathbf{VGra}$, is the subcategory of the arrow category $\mathbf{Gra}^{\to}$ consisting of covers. Any ordinary graph $G$ can be considered as a visual graph $Id_G\colon G \to G$ which gives rise to embedding $\mathbf{Gra}$ into $\mathbf{VGra}$.

The functor $cod\colon$ **VGra** $\to$ **Gra** is a left adjoint to the embedding above (due to that naming functions are covers), that is, **Gra** is reflective in **VGra**. $\diamondsuit$

**2.2.3 Definition.** A *(finitary) signature of (diagram) predicate markers over* **VGra** is defined to be a pair $\Pi = (Pred, Shp)$ with $Pred = \{P_1, \ldots, P_m\}$ a finite set of symbols called *predicate markers* and $Shp$ a function which assigns to each $P \in Pred$ a decidable set $Shp(P)$ of finite visual graphs called *arity shapes* of $P$.

Often we will use a shorter term **VGra**-signature for the construct above. $\diamondsuit$

The set $Shp(P)$ specifies the kind of diagrams on which the marker $P$ can be hung. For example, the arity of the relationship marker is the set of graphs which are finite sets of arrows with a common source, the arity of the identity marker is the set consisting of a single visual graph which is an arrow between two nodes with the same name (Table 1).

**2.2.4** By replacing in the definition above visual graphs by ordinary graphs, one comes to the definition of *predicate signature over* **Gra** (**Gra**-signature in short).

Given a signature $\Pi = (Pred, Shp)$ over **VGra**, by taking $D_N$ for each $D \in ShpP$ and each $P \in Pred$, one obtains a signature $\Pi^\star = (Pred, Shp^\star)$ over **Gra**. Conversely, any **Gra**-signature can be considered as a trivial **VGra**-signature in which naming in all shapes is identity. $\diamondsuit$

**2.2.5 Definition.** Let $\Pi = (Pred, Shp)$ be a **VGra**-signature.

(i) A *visual $\Pi$-sketch* is defined to be a tuple $\mathcal{S} = (G, P_1^{\mathcal{S}}, \ldots, P_m^{\mathcal{S}})$ where $G$ is a visual graph, *the carrier of* $\mathcal{S}$, and each $P_i^{\mathcal{S}}$ is a collection (maybe, empty) of diagrams in $G$ with shapes from $Shp(P_i)$, that is, $P_i^{\mathcal{S}} \subset \{\delta\colon D \to G \mid D \in Shp(P_i) \subset \mathbf{VGra}\}$. In addition, the visual component of any diagram $\delta \in P_i^{\mathcal{S}}$, $\delta_V$, must be monic. Diagrams from $P_i^{\mathcal{S}}$ are to be thought of as *marked* by $P_i$, and the monicity condition provides easy recognition of their shapes in the visual sketch. The graph $G$ will be also denoted by $|\mathcal{S}|$, sets $P_i^{\mathcal{S}}$ by $\mathcal{S}.P_i$ and $\mathcal{S}.\Pi$ will denote the family $(\mathcal{S}.P_1, \ldots, \mathcal{S}.P_m)$.

A $\Pi$-sketch $\mathcal{S}$ is called finite if $|\mathcal{S}|$ and all $\mathcal{S}.$ are finite.

(ii) A *visual sketch morphism* from $\mathcal{S}$ to $\mathcal{S}'$ is a morphism $h\colon G \to G'$ of underlying visual graphs compatible with marked diagrams: for any diagram $\delta$ in $G$ marked by $P_i$, $\delta \in \mathcal{S}.P$, the composition $\delta_V \blacktriangleright h_V$ must be monic (*ie*, the restriction of $h_V$ on the image of $\delta_V$ is injective) and the image of the diagram in $G'$ must be also marked my $P_i$, that is, $(\delta \blacktriangleright h) \in \mathcal{S}'.P$. This constitutes a category $\mathbf{VSke}_\Pi$ of all visual $\Pi$-sketches and mappings between them.

Morphism $h\colon \mathcal{S} \to \mathcal{S}'$ is called *inclusion* if $|\mathcal{S}| \subset |\mathcal{S}'|$ and $h$ is the corresponding inclusion. In this case we also say that $\mathcal{S}$ is a *subsketch* of $\mathcal{S}$', $\mathcal{S} \subset \mathcal{S}'$.

(iii) By replacing anywhere in the above definition "visual graph" on "ordinary graph" and removing the monicity reservations, one gets the definition of an *ordinary $\Pi$-sketch* (in a predicate signature $\Pi$ over **Gra**). $\diamondsuit$

**2.2.6** An example of visual sketch is shown on Fig. 5(a). One might ask why the visual diagram $P(a, b, c)$ is repeated twice. Indeed, while the repetition of the arrow $c$ is necessary to declare the constraint $\smile (c, a, c)$, the arrow $b$ in the triangle $\langle \mathtt{P} \rangle''$, and the marker $\langle \mathtt{P} \rangle''$ are logically redundant and might be removed. However, if the sketch in question is only a fragment of a larger sketch with several diagrams

adjoint to the logical-arrow triangle $(a, b, c)$, then it is very convenient to simplify the topology of the graph by repeating logical nodes and arrows as much as required. The simplest situation where diagram repetition is reasonable is when the shape consists of a single arrow or node. For example, drawing arrows $c$ twice with double-body marker is also logically redundant but if one arrow $c$ has double-body whereas the other is ordinary then the reader of the sketch might have doubts whether the marker on $c$ is a misprint. And indeed, it is known that a certain abundance of human languages is an effective mechanism of increasing reliability of communication. To summarize, the reasons justifying reiteration of logical sketch items in visual sketches are in cognitive psychology beyond mathematics but it is the matter of mathematics to formalize visual sketches and related machinery inasmuch as they are used. $\diamondsuit$

Thus, a visual sketch can contain a lot of logically inessential (but practically useful) details. The system should be capable to forget these details and extract from the visual sketch its logical core: what matters in semantics is the name graph and diagrams in it, *ie*, the *name sketch* quotient. However, an immediate output of the extraction is not an ordinary (name) sketch but an indexed (name) sketch.

**2.2.7 Definition** *(cf.Makkai[31])* Let $\Pi = (Pred, Shp)$ be a signature over **Gra**.

A *indexed* $\Pi$-sketch $\mathcal{S}$ is set up by the following data:

(a) an ordinary graph $G = |\mathcal{S}|$, the carrier of $\mathcal{S}$;

(b) for each predicate marker $P \in Pred$, some set $\mathcal{S}.P$ and a family,

$$\langle \delta^* \colon\ D \to G \mid \delta \in \mathcal{S}.P,\ D \in Shp(P) \rangle\ ,$$

of diagrams in $G$ indexed by $\delta \in \mathcal{S}.P$ and with shapes from $Shp(P)$.

Items of $G$ are called *names* and elements of the set $\mathcal{S}.P$ are called *visual diagrams*. Given $\delta \in \mathcal{S}.P$, $\delta^*$ is called the *name diagram* corresponding to $\delta$. This terminology is motivated by treating indexed sketches as constructs obtainable from visual sketches (construction 2.2.8 below).

A *morphism* of indexed sketches, $h\colon \mathcal{S} \to \mathcal{S}'$, is a graph morphism $|h|\colon |\mathcal{S}| \to |\mathcal{S}'|$ coupled with, for any $P \in Pred$, a mapping of visual diagrams $h\colon \mathcal{S}.P \to \mathcal{S}'.P$ s.t. their name diagrams are respected (*ie*, the square below commutes):

$$\begin{tikzcd}
\mathcal{S}.P \arrow[r, "*"] \arrow[d, "h"'] & \mathbf{Gra}(Shp(P), |\mathcal{S}|) \arrow[d, "-\triangleright |h|"] \\
\mathcal{S}'.P \arrow[r, "*'"] & \mathbf{Gra}(Shp(P), |\mathcal{S}'|)
\end{tikzcd}$$

This constitutes a category $\mathbf{ISke}_{\Pi}$ of indexed $\Pi$-sketches.

**2.2.8 Construction.** Let $\Pi$ be a signature over **VGra** and $\Pi^\star$ is its **Gra**-counterpart. Any visual $\Pi$-sketch $\mathcal{S}$ gives rise to an indexed $\Pi^\star$-sketch, $\mathcal{S}^\star$, as follows:

(i) if $|\mathcal{S}|$ is $G$, then $|\mathcal{S}^\star| = G_N$, *ie*, the carrier graph of $\mathcal{S}^\star$ is the name graph of the $|\mathcal{S}|$-carrier;

(ii) for each $P \in Pred$, $\mathcal{S}^\star.P \stackrel{\text{def}}{=} \mathcal{S}.P$ and

(iii) given $\delta\colon D \to G \in \mathcal{S}^\star.P$, $\delta^* \stackrel{\text{def}}{=} \delta_N\colon D_N \to G_N$

The construction gives rise to a functor $-^\star\colon \mathbf{VSke}_{\Pi} \to \mathbf{ISke}_{\Pi^\star}$. Conversely, any indexed sketch $\mathcal{S}$ gives rise to a visual sketch with the carrier $G\colon G_V \to G_N$ where $G_V$ is the disjoint union $|\mathcal{S}| \cup$

$\bigcup \langle dom(\delta^*) \mid \delta \in \mathcal{S}.P, P \in Pred \rangle$ and the naming mapping is evident; visual diagrams are also evident. This constitutes an embedding $j\colon \mathbf{ISke}_{\Pi^\star} \to \mathbf{VSke}_{\Pi}$. $\diamondsuit$

Example of transforming a visual sketch into an indexed one is presented on Fig. 5(b).

**2.2.9 Proposition.** The functor $(-)^\star$ is a left adjoint to $j$ and preserves monomorphisms. In its turn, $j$ preserves filtered colimits and is powerful, *ie*, induces bijections between subobject lattices of $\mathcal{S}$ and $j\mathcal{S}$. $\diamondsuit$

**2.2.10 Definition.** Two visual $\Pi$-sketches $\mathcal{S}_1, \mathcal{S}_2$ are *(logically) equivalent*, $\mathcal{S}_1 \simeq \mathcal{S}_2$ if indexed sketches hidden in them are isomorphic, $\mathcal{S}_1^\star \cong \mathcal{S}_2^\star$. $\diamondsuit$

**2.2.11 Remark.** Categories $\mathbf{ISke}_{\Pi^\star}$ and $\mathbf{OSke}_{\Pi^\star}$ are exactly (with minor reservations) the two sketch categories invented by Makkai ([31]). Indeed, it is easy to see that definition 2.2.3 and definition 2.2.5 without the monicity condition can be formulated for any base category $\mathbf{G}$ instead of $\mathbf{Gra}$ provided it has a good notion of finiteness, eg, if $\mathbf{G}$ is finite topos or, at least, is locally finitely presentable (lfp). Thus, the notions of signature $\Pi$ over lfp category $\mathbf{G}$, a(n indexed) $\Pi$-sketch and the categories of indexed and ordinary $\Pi$-sketches are defined. Makkai designates these categories by $\mathbf{G}||\Pi$ and $\mathbf{G}|\Pi$ respectively. The only distinction between our $\mathbf{ISke}_{\Pi}$,$\mathbf{OSke}_{\Pi}$ and Makkai's $\mathbf{G}||\Pi$, $\mathbf{G}|\Pi$ is that in our setting $Shp(P)$ is a set of $\mathbf{G}$-objects rather than a single object. Of course, we can reduce our setting to Makkai's one by splitting $P$ into a family of markers $\langle P_D \mid D \in Shp(P) \rangle$ with $Shp(P_D) = D$. So, for any signature $\Pi$ over $\mathbf{Gra}$, $\mathbf{OSke}_{\Pi} = \mathbf{Gra}|\Pi_0$, $\mathbf{ISke}_{\Pi} = \mathbf{Gra}||\Pi_0$ where $\Pi_0$ denotes the result of splitting $\Pi$. Note, however, that in practically interesting cases (where the predicate of being jointly monic is a must) the set $Pred_0$ is infinite even though the original set $Pred$ is finite! As for visual sketches, due to the monicity condition, $\mathbf{VSke}_{\Pi}$ is a proper subcategory of $\mathbf{VGra}|\Pi_0$. $\diamondsuit$

Thus, given a visual signature $\Pi$, one has three related categories $\mathbf{VSke}_{\Pi}$, $\mathbf{ISke}_{\Pi^\star}$, $\mathbf{OSke}_{\Pi^\star}$ connected by adjunctions as shown in equation (1) (the right adjunction was built by Makkai). With minor reservations, proofs and results of Makkai [31] are applicable in our setting and the following result can be conjectured.

**2.2.12 Proposition?** For any predicate signature $\Pi$ over $\mathbf{VGra}$, categories of visual and indexed sketches, $\mathbf{VSke}_{\Pi}$ and $\mathbf{ISke}_{\Pi^\star}$, are presheaf toposes whose finite objects are sketches carried by finite graphs and containing a finite number of (visual) diagrams. As for ordinary sketches, $\mathbf{OSke}_{\Pi^\star}$ is a full regular-epi-reflective subcategory of $\mathbf{ISke}_{\Pi^\star}$.

*Instead of proof.* The part with $\mathbf{ISke}_{\Pi^\star}$ and $\mathbf{OSke}_{\Pi^\star}$ is proven by Makkai. The part with $\mathbf{VSke}_{\Pi}$ seems can be treated in parallel with Makkai's proof but in the struggle with the deadline the author had no time to check details. $\diamondsuit$

Even if the above proposition fails, it is clear that some weaker version holds, at any rate, $\mathbf{VSke}_{\Pi}$ is lfp category. An important practical consequence is that one can perform with sketches a lot of operations. The more so if the proposition holds: then graph-based counterparts of all set-theoretical operations are definable. That is, the treatment of data specifications by generalized sketches provides an extremely rich algebraic framework for manipulations with specifications (cf. [26]).

### 2.3 Formal semantics for sketches

A predicate signature itself is a purely syntactical notion. It can be of interest if only some semantics of predicate markers is presupposed. That is, each marker $P_i$ should be interpreted as a certain property $[\![ P_i ]\!]$ of set-and-function diagrams of shapes from $Shp(P_i)$.

Let $\mathcal{S} = (G, P_1^{\mathcal{S}}, \ldots, P_m^{\mathcal{S}})$ be a visual sketch. Intuitively, an extent of $\mathcal{S}$ is a mapping $\varepsilon$ which assigns (finite) sets to nodes of $G_N$ and functions between them to arrows of $G_N$. In addition, if some diagram $\delta$ in $G$ is marked by $P_i$, $\delta \in P_i^{\mathcal{S}}$, then the corresponding diagram of sets and functions, $\varepsilon(\delta) = \delta_N \Rrightarrow \varepsilon$, must satisfy the predicate $[\![ P_i ]\!]$. Thus, each pair $(P_i, \delta)$ with $\delta \in P_i^{\mathcal{S}}$ can be thought of as a constraint to a possible extent of $|\mathcal{S}|_N$. To formalize this description we proceed as follows.

**2.3.1 Construction.**

**(i)** We assume some universe of proto-objects (urelements) $Obj$ is given, and let $U = \mathbf{Set}(Obj)$ denote the graph whose nodes are finite subsets of $Obj$ and arrows are functions between them so that an extent of a given sketch $\mathcal{S}$ is a graph morphism $\varepsilon\colon |\mathcal{S}|_N \to U$. Of course, to provide sufficient richness of the graph $U$ one should presuppose that the universe $Obj$ possesses some closure properties: closedness under hereditary finite sets, $Obj = \mathbf{HF}(Obj)$, would be sufficient (cf. [40, 36]).

**(ii)** *Semantics* of a marker $P$ is defined to be a mapping

$$[\![ P ]\!]\colon \mathbf{Gra}(Shp(P), U) \to \{true, false\}.$$

In addition, if one makes a reasonable assumption that urelements are uninterpreted abstract objects, then diagram predicates should not depend on their permutations:

$$\text{(GEN)} \qquad [\![ P ]\!](\pi^+\delta) = [\![ P ]\!](\delta)$$

for any permutation $\pi \in Aut(Obj)$ and any diagram $\delta\colon D \to U, D \in Shp(P)$, here $\pi^+\delta$ denotes the action of $\pi$ on the diagram $\delta$. This equation is a predicate analog of the well-known genericity condition for queries (introduced by Chandra and Harel [11], and well known in the database theory).

**(iii)** Let $G\colon G_V \to G_N$ be a visual graph, $P$ a predicate marker with visual shapes and $\delta$ some visual diagram in $G$ of shape from $Shp(P)$. A pair $(P, \delta)$ is called *a constraint*, and an extent of the name graph, $\varepsilon\colon G_N \to U$, is said to *satisfy* it if $[\![ P ]\!](\delta_N \Rrightarrow \varepsilon) = true$. $\diamondsuit$

Thus, a visual sketch $\mathcal{S}$ can be thought of as a complex specification which comprises (i) a structure specification – the name graph $|\mathcal{S}|_N$, and (ii) a set of constraints – the set $\mathcal{S}^{\star\bullet}.\Pi = \{(P, \delta_N) : P \in Pred, \delta \in \mathcal{S}.P\}$.

The considerations above can be expressed in a more sketch-oriented manner.

**2.3.2 Construction.** Mappings $[\![ P_i ]\!]$, $i = 1, \ldots, m$ convert the graph $U$ into an ordinary sketch $\mathcal{U} = (\mathcal{U}, P_1^{\mathcal{U}}, \ldots, P_m^{\mathcal{U}})$ with $P_i^{\mathcal{U}} = [\![ P_i ]\!]^{-1}(true)$. Genericity means that $\pi^+\delta \in P_i^{\mathcal{U}}$ for any $\delta \in P_i^{\mathcal{U}}$, that is, any permutation $\pi^+$ is an automorphism of the sketch $\mathcal{U}$, $\pi^+ \in Aut(\mathcal{U})$.

Now it is easy to see that for a given visual sketch $\mathcal{S} = (G, P_1^{\mathcal{S}}, \ldots, P_m^{\mathcal{S}})$, an extent mapping $\varepsilon\colon G_N \to U$ satisfies all the diagram constraints in $\mathcal{S}.\Pi$ iff $\varepsilon$ is a sketch morphism $\varepsilon\colon \mathcal{S}^{\star\bullet} \to \mathcal{U}$. $\diamondsuit$

## 3 Databases as graphical algebras: Data derivation via diagram operations

As it was mentioned in introduction, data constraints involving queries (data operations) should appear in semantic schemas much more often than it is assumed in the literature. Correspondingly, sketches employed as semantic schemas should contain diagrams labeled by operation markers.

### 3.1 Operational sketches in database design

#### Examples

**3.1.1** Consider, again, the simple universe described on Fig. 4. An important constraint that should be added to the schema is the unique identification of any *currently* married couple by either its husband or its wife. In other words, the subset of the relation *Married* for which the value of the attribute '*ddate*' is $-$ must be of the one-one relationship type. To express such a constraint, one should, first of all, extend the original sketch with the corresponding derived items, that is, define new nodes and arrows (see Fig. 6) as specified below in an *ad hoc* syntax:

$$\begin{array}{l} \texttt{define}\ (CurrMarried^*, cm^*, dd^*)\ \texttt{as}\ \mathbf{CoImage}\ (\{-\}, -, ddate) \\ \texttt{define}\ (husb^\circ)\ \texttt{as}\ \mathbf{Composition}(cm^*, husb) \\ \texttt{define}\ (wife^\circ)\ \texttt{as}\ \mathbf{Composition}(cm^*, wife). \end{array} \tag{2}$$

Here bold names refer to some predefined operations on sets and functions. The peculiarity of these operations is that their input/output data are specified by graphs which, moreover, carry sketch structures specifying pre/postconditions for data (for example, in the input data for **CoImage** (Table 2) one arrow is marked by the IsA marker). Actually, these operations are *diagram operations* over sketches: a diagram operation $F$ is specified by a sketch $\mathcal{D}_F$ denoting its interface in which a subsketch $\mathcal{D}_F^{\text{in}}$ of input data is designated, that is, an operation $F$ is specified by an inclusion $\iota_F : \mathcal{D}_F^{\text{in}} \hookrightarrow \mathcal{D}_F$ (in the Table 2 sketches $\mathcal{D}_F$ are called output sketches). The body of operation is then a procedure $[\![ F ]\!]$ which calculates an extension of derived items of $\mathcal{D}_F$ from a given extension of $\mathcal{D}_F^{\text{in}}$.

Of course, the string-based notation (2) is not appropriate for diagram operations. It is more suggestive to use graphical notation as shown on Fig. 6. In order to distinguish basic items from the derived ones, we agree to denote the latter by hanging various superscripts (like $', *, \circ$ *etc*) on operation markers and then to label the derived items produced by an operation by the same superscript. Basic nodes are additionally distinguished by filling-in with small dots intended to remind about the extent to be stored while derived nodes are transparent; value domain nodes are similar to derived classes in that their extension is also specified rather than stored and so they are transparent too. $\diamondsuit$

**3.1.2** It is important to distinguish the origin of the IsA-marker that labels the arrow $cm^*$ from that of the monic markers labeling arrows $husb^\circ$ and $wife^\circ$. The IsA-marker on $cm^*$ is a postcondition of the operation *CoImage*: the corresponding constraint should be satisfied automatically provided the input data satisfy the input sketch. There are similar automatically satisfied postcondition markers in specifications of other operations: such a marker is present in the output sketch of operation (see Table 2). In contrast, monic markers hung on the arrows $wife^\circ, husb^\circ$ express the actual constraint arising from

semantics of the schema, *a priori* these arrows should not be monic. This is just an instance of a general phenomenon when some constraints to a schema involve queries against it (cf. [29]). ♢

**3.1.3** An important class of constraints involving queries are those where equality of two queries is stated. Such constraints are quite naturally and often occur in schemas resulted from view integration. The following toy example demonstrates the phenomenon.

Two views on the same universe are specified by sketches on Fig. 7(a). Assume it is somehow known that mail persons of the left view are exactly unmarried men of the right view. To specify this constraint we (i) augment sketches with derived items as shown on Fig. 7(b), and (ii) state the *correspondence equation* $\overline{S_1}.MPerson^* = \overline{S_2}.UnMMan''$. The global schema is then obtained by disjoint merging sketches and factorizing the merge by the equation, the result is presented on Fig. 7(c). This sketch specifies both local schemas and, in addition, a global equational constraint connected with the node $UnMMan^{*''}$ (notice two superscripts indicating that the node occurs into derived items of two operations). ♢

**Discussion**

**3.1.4** While relational query languages are conceptually quite clear, there was a considerable debate in the DB theory literature on the nature of queries in the OO databases ([1, 28, 37, 3] to mention few publications). Our analysis has shown that a major part of discussion is caused not by the problem itself but rather by the absense of suitable language: no general specification pattern for queries against OODB is known to the community. In contrast, the framework of diagram operations over sketches provides a great flexibility and captures

(i) *class-preserving* queries, when only arrows are derived (as, *eg*, in the operation of composition),

(ii) *object-preserving* queries, when every new derived class is a subclass of some input class (*eg*, in the CoImage operation),

(iii) *object-creating* queries, when at least one derived class is not a subclass of an input class (*eg*, the Pull-Back operation).

Moreover, the same pattern of diagram operations captures

(iv) constraint (predicate) inference when $|\mathcal{D}_F^{\text{in}}| = |\mathcal{D}_F|$ and the only difference between these sketches is in the presence of additional derived markers in $\mathcal{D}_F$ (see, *eg*, the two last rows of Table 2).

In this case it would be suggestive to call diagram operations *diagram inference rules*. For example, the anti-commutativity marker in the sketch on Fig. 4 is derived by an application of the operation **Rule_1**, the marker of this operation is not shown on the picture (but, of course, occurs in the logical sketch behind it). ♢

**3.1.5 Remark on notation.** It would be suggestive to depict basic nodes/arrows and operation markers by one color, say, black, derived items (nodes, arrows, markers) by another color, say, green, and predicate markers denoting actual (not derived) constraints by, say, red. Our practice of applying sketches shows that colored sketches are quite readable despite a large number of conventional signs. Unfortunately, the black-white framework enforces to use less evident means and, as was said, we agree to designate the green color of item by some superscript near its name. This does not work for various icon-markers (arcs on arrow sources, figure tails and heads of arrows *etc*)

and we will consider them green by default while red icon-markers are additionally framed by a small circle (not to be confused with the superscript $\circ$).

Thus, in the sketch $\overline{\mathcal{S}}$ on Fig. 6, the node *CurrMarried* and arrows $husb^\circ, cm^*$, $dd^*$, $wife^\circ$ are green but small tail arcs on $husb^\circ$, $wife^\circ$ are red. The marker $\langle\neq\rangle$ is green whereas the arc spanning the triple $(husb, wife, mdate)$, the arc spanning the pair $(m, w)$, the head of the arrow *wife* and the tail of *ssn* are red. Other items are black. To make the sketch more understandable the reader is encouraged to paint it! $\diamondsuit$

**3.1.6** The pattern of schema integration exhibited in section 3.1.3 works in more complex situations as well: the initial sketches are augmented via applying diagram operations from a predefined set, a set of correspondence equations determines a span between them and, finally, the global schema is obtained by taking push-out of the span. It is remarkable that CT provides a clear, brief and effective description of a problem which seems extremely difficult with *ad hoc* approaches – an abundance of them was developed (and is continuing to grow) in the database theory literature.[3]

Sketch integration is nothing but a graphical counterpart of the well known way of specifying ordinary algebras by generators and defining relations between them (explained in any textbook on universal algebra). The only difference is that generators are organized into a graph and equations are stated for diagram operation terms. Thus, with sketches one obtains a comprehensible and compact notational system extremely convenient for managing schema integration (more complex examples were demonstrated in [8, 10]). $\diamondsuit$

## 3.2 Formalizing operational sketches

### Preliminary description

**3.2.1** Sketches we considered above contain diagrams marked by predicate and operation markers. In addition, it can occur that input data for an operation are specified by a diagram of sets and functions subjected to certain conditions. Thus, shapes of operation markers are not graphs but sketches. Moreover, input data for some operation may be constrained by equations involving other, preceding, operations. For example, the shape of the operation of universal arrow of pull-back, *UAPB*, is specified by two squares and one of them should be pull-back while the other must be commutative (Table 2). Thence, a signature of diagram operations should be partially ordered into a hierarchy: for a marker $F$ a finite set $F^-$ of its predecessors is defined and the shapes of $F$ are sketches which can involve markers from $F^-$.

To simplify presentation, we will be dealing with the case of unordered signature when all operation markers are independent, that is, for each operation marker $F$ the set of its predecessors $F^-$ does not contain operation markers and equals to some predefined signature of predicate markers, $\Pi$. Transition to the general case is not very difficult[4] . $\diamondsuit$

[3] As a curious case we mention the paper [5] where an attempt was made to provide the schema integration operation with denotational semantics but, as a result, it proved non-associative (!); to achieve associativity the authors were enforced to introduce intricate *ad hoc* complications into their formalism.

[4] in syntax, but in semantics, monadicity of the forgetful functor from the category of algebras into the category of $\Pi$-sketches can be lost in the general situation (cf.[2])

**3.2.2** By forgetting input subsketches, operation markers can be considered as predicate markers and thus graphical images on Fig. 6,7 are sketches like those considered before in section 2. However, these new *operational* sketches have two peculiarities: (a) some of markers they involve are richer than predicate markers in that they contain input subsketches, (b) items of operational sketches are partitioned into basic and derived.

On one hand, the new structure provides new specification capabilities (*eg*, equational constraints). On the other hand, it imposes additional syntactical restrictions: if $\Sigma$ is a joint signature of predicate and operation markers and $\Sigma_{pr}$ is its reduct where input subsketches of shape sketches are removed, then not every well-formed $\Sigma_{pr}$-sketch is also a well-formed operational $\Sigma$-sketch. The latter must satisfy the following additional conditions:

(i) if an item is designated as derived, it must occur in the output items of at least one diagram labeled by an operation markers,

(ii) each input item of such a diagram must be either basic or be an output item of some preceding diagram labeled by operation marker.

The latter condition induces a tree-like structure on operation diagrams embodied into the sketch similar to the tree-like representation of terms in the ordinary algebra. Thus, parsing of operational sketches is required in order to check whether they are well-formed. $\diamondsuit$

**Signatures and algebras**

Let $\Pi$ be a predicate signature over **VGra**.

**3.2.3 Definition.** A *signature of diagram operations* over $\Pi$ is a couple $\Phi = (Op, Shp)$ with $Op = (F_1, \ldots, F_n)$ a finite set of *operation markers* and $Shp$ a mapping which assigns to each marker $F_j$ a decidable set $Shp(F_j)$ of inclusions between finite visual $\Pi$-sketches called *arity shapes* of $F_j$. That is, given $F \in Op$, an element of $Shp(F)$ is an inclusion arrow $\iota : \mathcal{D}^{\text{in}} \hookrightarrow \mathcal{D}$ for which $\mathcal{D}^{\text{in}}$ is interpreted as a sketch specifying possible input data for the operation $F$ and $\mathcal{D}_F$ specifies the entire interface. To simplify notation we will further assume that $Shp(F)$ consists of a single inclusion arrow $\iota_F : \mathcal{D}_F^{\text{in}} \hookrightarrow \mathcal{D}_F$. The changes necessary to conform with the general situation will be always evident. $\diamondsuit$

**3.2.4 Definition.**

(i) A *$\Phi$-algebra* $\mathcal{A}$ is defined to be an ordinary $\Pi^\star$-sketch $\mathcal{A}_\Pi = (G, P_1^{\mathcal{S}}, \ldots, P_m^{\mathcal{S}})$ which, in addition, carries the operations from $\Phi$, that is, for any $F \in Op$ there is defined a mapping

$$F^{\mathcal{A}} : \mathbf{Ske}_{\Pi^\star}(\mathcal{D}_F^{\text{in}^{\star\bullet}}, \mathcal{A}_\Pi) \to \mathbf{Ske}_{\Pi^\star}(\mathcal{D}_F{}^{\star\bullet}, \mathcal{A}_\Pi)$$

where, we remind, $\mathcal{S}^{\star\bullet}$ is the ordinary (name) $\Pi^\star$-sketch extracted from a visual $\Pi$-sketch $\mathcal{S}$. In addition, $\iota_F \Rrightarrow F^{\mathcal{A}}(e) = e$ for any $e \in \mathbf{Ske}_{\Pi^\star}(\mathcal{D}_F^{\text{in}^{\star\bullet}}, \mathcal{A}_\Pi)$, which guarantees projection of the input data.

Thus, $\Phi$-algebra is a tuple $\mathcal{A} = (\mathcal{A}_\Pi, F_1^{\mathcal{A}}, \ldots, F_n^{\mathcal{A}})$ with $\mathcal{A}_\Pi$ a carrier $\Pi$-sketch, and $F_j^{\mathcal{A}}$ ( $j = 1, \ldots, n$ ) are functions as above. To simplify notation, sometimes we will not point out the functor $\_^{\star\bullet}$ explicitly, and write $\Pi$ for $\Pi^\star$ as well. The intended meaning will be always recoverable from the context.

(ii) Given two $\Phi$-algebras $\mathcal{A}', \mathcal{A}''$, *a morphism* from $\mathcal{A}'$ to $\mathcal{A}''$ is defined to be a $\Pi$-sketch morphism $h : \mathcal{A}'_\Pi \to \mathcal{A}''_\Pi$ s.t. for any operation marker $F \in Op$ the following diagram commutes:

$$\begin{array}{ccc}
\mathbf{Ske}_{\Pi}(\mathcal{D}_F^{\mathrm{in}}, \mathcal{A}'_{\Pi}) & \xrightarrow{F^{\mathcal{A}'}} & \mathbf{Ske}_{\Pi}(\mathcal{D}_F, \mathcal{A}'_{\Pi}) \\
\Big\downarrow {\scriptstyle _\!\!\triangleright\!\!\triangleright h} & & \Big\downarrow {\scriptstyle _\!\!\triangleright\!\!\triangleright h} \\
\mathbf{Ske}_{\Pi}(\mathcal{D}_F^{\mathrm{in}}, \mathcal{A}''_{\Pi}) & \xrightarrow{F^{\mathcal{A}''}} & \mathbf{Ske}_{\Pi}(\mathcal{D}_F, \mathcal{A}''_{\Pi})
\end{array}$$

This constitutes the category $\mathbf{Alg}_{\Phi}\Pi$ of *$\Phi$-algebras over $\Pi$*.

(iii) Let $\Sigma$ denotes the merge of predicate and operation signatures into one signature $\Sigma = (Pred, Op, Shp)$ over **VGra** and $\Sigma_{pr}$ denotes its reduct where input subsketches of $Op$-markers shapes are forgotten. $\Sigma_{pr}$-sketchs will be also called *$\Sigma$-sketchs* and $\Phi$-algebras over $\Pi$ will be called *$\Sigma$-algebras*, $\mathbf{Alg}_{\Sigma}$ is then the same as $\mathbf{Alg}_{\Phi}\Pi$. Note, any $\Sigma$-algebra is a $\Sigma$-sketch but not the reverse. $\diamondsuit$

**Semantics**

The notion of $\Phi$-algebra gives an algebraic semantics for operation markers. The concrete (set-theoretical) semantics is its special case: for a marker $F$, as soon as data (sets and functions) structured according to $\mathcal{D}_F^{\mathrm{in}}$ are given, then some procedure $[\![ F ]\!]$ assigned to $F$ compute data structured according to $\mathcal{D}_F$ s.t. the input data are protected. In other words, the arity of an operation marker can be considered as a sketch-based specification of some procedure operating sets and functions.

**3.2.5 Construction.** Let $\mathcal{U}_{\Pi}$ denotes the semantic $\Pi$-sketch introduced above in construction 2.3.1.

*Semantics* of a marker $F$ with arity shape $\iota_F\colon \mathcal{D}_F^{\mathrm{in}} \hookrightarrow \mathcal{D}_F$ is defined to be a mapping

$$[\![ F ]\!]\colon \mathbf{Ske}_{\Pi}(\mathcal{D}_F^{\mathrm{in}}, \mathcal{U}_{\Pi}) \to \mathbf{Ske}_{\Pi}(\mathcal{D}_F, \mathcal{U}_{\Pi}).$$

s.t. the input data are preserved as above and, in addition, the following *genericity condition*

(GEN) $$[\![ F ]\!](\pi^+\delta) = \pi^+([\![ F ]\!]\delta) \text{ for any } \delta\colon \mathcal{D}_F^{\mathrm{in}} \to \mathcal{U}_{\Pi}, \pi \in Aut(Obj).$$

is satisfied (if, of course, one makes the usual assumption that urelements are uninterpreted abstract objects).

Mappings $[\![ F ]\!]$ convert the sketch $\mathcal{U}_{\Pi}$ into a $\Sigma$-algebra

$$\mathcal{U} = (\mathcal{U}_{\Pi}, P_1^{\mathcal{U}}, \ldots, P_m^{\mathcal{U}}, F_1^{\mathcal{U}}, \ldots, F_n^{\mathcal{U}})$$

with $P_i^{\mathcal{U}} = [\![ P_i ]\!]$ as above and $F_j^{\mathcal{U}} = [\![ F_j ]\!]$. Genericity means that any action $\pi^+$ is an automorphism of the algebra $\mathcal{U}$, $\pi^+ \in Aut_{\Sigma}(\mathcal{U})$. $\diamondsuit$

Thus, a signature of predicate and operation markers coupled with their intended semantics forms a stage for graph-based logic and model theory. A categorist would call such a stage a *(logical) doctrine*. The DB context suggests another name – *data definition language (DDL)*.

**3.2.6 Definition.** A *data definition language* is a triple $\mathcal{L} = (\Pi, \Phi, [\![\ ]\!])$ with $\Pi$ a predicate signature, $\Phi$ an operation signature over $\Pi$ and $[\![\ ]\!]$ a mapping assigning semantic meaning to markers as explained above. $\diamondsuit$

### Query mechanism

**3.2.7** An application of an operation $F$ to a visual $\Pi$-sketch $\mathcal{S}$ is set by a sketch morphism $q\colon \mathcal{D}_F^{\mathrm{in}} \to \mathcal{S}$. For instance, in the example of Fig. 6(a) the application of the $\langle \mathtt{CoIm} \rangle$-operation is determined by the mapping presented in the table below on the left of the double line:

| $\mathcal{D}^{\mathrm{in}}_{\langle \mathtt{CoIm} \rangle}$ | $X$ | $Y$ | $f$ | $B$ | $b$ | $B'$ | $b'$ | $f'$ |
|---|---|---|---|---|---|---|---|---|
| $\mathcal{S}$ | $Married$ | $Date$ | $ddate$ | $\{-\}$ | $-$ | $CurrMarried^*$ | $cm^*$ | $dd^*$ |

Semantically, $q$ shows which sets and functions from the extent of $\mathcal{S}$ should be passed to the input of the procedure $F$. So, a pair $Q = (F, q)$ is defined to be a *query* against $\mathcal{S}$, and if $e\colon \mathcal{S}^{\star\bullet} \to \mathcal{U}$ is an extent of $\mathcal{S}$, the answer is given by the extent $[\![ F ]\!]\,(q^{\star\bullet} \Rrightarrow e)$ of $\mathcal{D}_F$ (see the outer square on left diagram below, it is in the category $\mathbf{OSke}_{\Pi^\star}$).

$$\begin{tikzcd}
\mathcal{D}_F^{\mathrm{in}} \arrow[r, hook, "\iota_F"] \arrow[d, "q"'] \arrow[dr, phantom, "\langle \mathtt{PO} \rangle"] & \mathcal{D}_F \arrow[d, "q'"] \arrow[ddr, "{[\![ F ]\!]\,(q \Rrightarrow e)}"] & \\
\mathcal{S} \arrow[r, hook, "\iota'_F"] \arrow[drr, "e"'] & Q(\mathcal{S}) \arrow[dr, "Q(e)"] & \\
& & \mathcal{U}
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathcal{D}_F^{\mathrm{in}} \arrow[r, hook, "\iota_F"] \arrow[d, "q"'] \arrow[dr, phantom, "\langle \mathtt{PO} \rangle"] & \mathcal{D}_F \arrow[d, "q'"] \\
\mathcal{S} \arrow[r, hook, "\iota'_F"] & Q(\mathcal{S})
\end{tikzcd}$$

Syntactically, the mapping $q$ allows to augment $\mathcal{S}$ with derived items of $\mathcal{D}_F$ via the push-out operation and so an extended sketch $Q(\mathcal{S})$ is obtained (see the right diagram, it is in the category $\mathbf{VSke}_{\Pi}$). Since the functor $(-)^{\star\bullet}$ is a left adjoint and hence preserves colimits, the push-out square on the right is carried into a push-out square on the left (the inner square in the left diagram). The universal property of push-outs is then provides a unique extent $Q(e)$ of derived items.

### Operational sketches and their parsing

A difficult problem with diagram operations is to find a proper notion of complex term composed from several operations. In the ordinary algebra composition of terms is easy to specify since input/output arities are sets. In contrast, input/output arities of diagram operations are complex sketch structures which may be connected (composed) in a diversity of ways. Description of a single act of composition is easy, it is given by the push-out operation as explained in 3.2.7. The problem we discuss is how to specify results of several consequent applications of push-outs.

A natural idea (first proposed in [16] and announced in [15]) is to treat diagram terms as trees labeled in a special way. The sample we should have in mind is the tree-based description of terms in the ordinary algebra where composed terms are built with operation symbols and variables taken from a predefined pool. The diagram version is as follows.

Let $\Sigma = (Pred, Op, Shp)$ be a signature of predicate and operation markers over $\mathbf{VGra}$.

**3.2.8 Construction.**

(i) An operation marker $C$ is called a *constant* if $\iota_C = \emptyset\colon \emptyset \hookrightarrow \mathcal{D}_C$ with $\mathcal{D}_C$ a $\Pi$-sketch. Each visual $\Pi$-sketch $\mathcal{S}$ determines a constant $C_{\mathcal{S}}$ with $\mathcal{D}_{C_{\mathcal{S}}} = \mathcal{S}$.

(ii) We fix a countable pool of *names* organized into a graph $Nms$ of node names and arrow names. To any finite visual $\Pi$-sketch $\mathcal{X}$ whose carrier name graph is a subgraph of $Nms$ we assign a constant operation marker $C_{\mathcal{X}}$ with $\mathcal{D}_{C_{\mathcal{X}}} = \mathcal{X}$, $\mathcal{D}^{\mathrm{in}}_{C_{\mathcal{X}}} = \emptyset$. Let

$$Base \stackrel{\mathrm{def}}{=} \{C_{\mathcal{X}} \mid \mathcal{X} \text{ is a finite } \Pi\text{-sketch and } |\mathcal{X}|_N \text{ is a subgraph of } Nms\}\,.$$

(iii) Each predicate marker $P \in Pred$ can be considered as an operation marker $P^+$ with shape $\iota_{P^+}\colon D_P \hookrightarrow \mathcal{D}_P$ where $D_P$ is the shape (visual) graph of $P$ and $\mathcal{D}_P$ is a $\Pi$-sketch with $|\mathcal{D}_P| = D_P$ and a single diagram $1_{D_P}\colon D_P \to D_P$. This operation hangs the marker $P$ on the graph $D_P$. Let $Pred^+$ denotes the set $\{P^+ \mid P \in Pred\}$.

(iv) Finally, we set $Op^+ \stackrel{\mathrm{def}}{=} Op \cup Base \cup Pred^+$ $\diamondsuit$

**3.2.9 Definition.** A $(\Sigma, Nms)$*-tree* is a pair $\mathcal{T} = (T, \lambda)$ where $T$ is a finite multi-rooted tree and $\lambda$ is a labeling functions which assigns to each node $k \in T$ a pair $(F_k, \delta_k)$ with $F_k \in Op^+$ and $\delta_k\colon |\mathcal{D}_{F_k}|_N \to Nms$ an ordinary graph morphism. $\diamondsuit$

**3.2.10 Notation.** With any node $k$ of $(\Sigma, Nms)$-tree there are correlated the following two commutative diagram displaying the notation we use:

$$\begin{tikzcd}
|\mathcal{D}_{F_k}|_V \arrow[r, "|\mathcal{D}_{F_k}|"] & |\mathcal{D}_{F_k}|_N \arrow[r, "\delta_k"] & Nms \arrow[d, equal] \\
|\mathcal{D}^{\mathrm{in}}_{F_k}|_V \arrow[r, "|\mathcal{D}^{\mathrm{in}}_{F_k}|"] \arrow[u, hook, "|\iota_{F_k}|_V"] & |\mathcal{D}^{\mathrm{in}}_{F_k}|_N \arrow[r, "\delta^{\mathrm{in}}_k"] \arrow[u, hook, "|\iota_{F_k}|_N"] & Nms
\end{tikzcd}
\qquad
\begin{tikzcd}
Out_V(F_k) \arrow[r, "Out(F_k)"] \arrow[d, hook, "I_V(F_k)"] & Out_N(F_k) \arrow[r, "|\delta_k|^{\mathrm{Out}}"] \arrow[d, hook, "I_N(F_k)"] & Nms_k^{\mathrm{Out}} \arrow[d, hook] \\
||\mathcal{D}_{F_k}||_V \arrow[r, "||\mathcal{D}_{F_k}||"] & ||\mathcal{D}_{F_k}||_N \arrow[r, "|\delta_k|"] & Nms_k \\
||\mathcal{D}^{\mathrm{in}}_{F_k}||_V \arrow[r, "||\mathcal{D}^{\mathrm{in}}_{F_k}||"] \arrow[u, hook, "||\iota_{F_k}||_V"] & ||\mathcal{D}^{\mathrm{in}}_{F_k}||_N \arrow[r, "|\delta^{\mathrm{in}}_k|"] \arrow[u, hook, "||\iota_{F_k}||_N"] & Nms_k^{\mathrm{in}} \arrow[u, hook]
\end{tikzcd}$$

The left diagram is in the category of graphs **Gra**, the right one is in the category of sets and obtained by considering graphs as plain sets of their items (arrows and nodes): given a graph $G$, $|G| \stackrel{\mathrm{def}}{=} NdsG \cup ArrsG$. $Out_V(F_k), Out_N(F_k)$ are complements of $||\mathcal{D}^{\mathrm{in}}_{F_k}||_V, ||\mathcal{D}^{\mathrm{in}}_{F_k}||_N$ respectively. $\diamondsuit$

Actually, each node of a tree denotes a computation (query) whose input should be taken from outputs of computations denoted by preceding nodes. The very data transferring between these node-comptutations is provided by common names in the ranges of the corresponding diagrams.

**3.2.11 Definition.** A $(\Sigma, Nms)$-tree is called *well-formed* if for each node $k$ the following condition is fulfilled:

(WFT) $\qquad Nms_k^{\mathrm{in}} \subset \bigcup \{Nms_j^{\mathrm{Out}} \mid j \in k^-\}$ and $Nms_k^{\mathrm{in}} \cap Nms_j^{\mathrm{Out}} \neq \emptyset$ for any $j \in k^-$,

here $k^-$ denotes the set of nodes preceding $k$. In particular, a node $k$ is a leaf iff $F_k$ is a constants. $\diamondsuit$

**3.2.12 Description.** $(\Sigma, Nms)$-trees provide a very flexible notational mechanism which we will describe briefly and informally.

Leaves are nodes of rank 0, jointly they denote the basic graph of names (generators) occurring into the specification. A root $r$ of rank 1 denotes either some simple diagram term composed from basic items (or signature constants) when $F_r \in Op$, or an assertion about basic items when $F_r = P^+ \in Pred^+$. In the latter case one can safely merge the leaves connected with the node $r$ into one leaf sketch and then add to it a diagram marked by $P$ (this is just the effect of operation $P^+$). If the rank of root $r$ is greater than 1 then it denotes either a complex term composed from items derived in preceding nodes when $F_r \in Op$, or an assertion about derived items when $F_r = P^+ \in Pred^+$ (see, *eg*, monic markers on arrows $husb^\circ$, $wife^\circ$ in Fig. 6).

A root is called *free* if $|\delta_r|^{\mathrm{Out}}$ is one-one and $Nms_r^{\mathrm{Out}}$ does not meet $Nms_k^{\mathrm{Out}}$ for any other node $k \in T$. Otherwise, some set of equations between (components of) terms is associated with the root $r$. In addition, if there are equations in the root $r'$ of some subtree $T'$ preceding $r$, then equations and $\Pi$-assertions associated with $r$ become conditional and the term of $r$ becomes partially defined (but its domain is defined by equations among other terms).

A tree is called *free* if any non-leaf node is a free root in the corresponding subtree. $\diamondsuit$

In the ordinary algebra composed terms are (intrinsically trees but externally) presented in some notational system: with or without brackets, prefix, infix *etc*. Sketches with operation markers we considered above in our toy SM examples are nothing but such visual presentations of $(\Sigma, Nms)$-trees which the database designer has in mind when he draws sketches. The tree structure is explicated by a special parsing procedure applicable to operational sketches. Briefly, it is as follows.

**3.2.13 Construction.** Any $(\Sigma, Nms)$-tree generates a $\Sigma$-sketch $\mathcal{S} = Ske_\Sigma(\mathcal{T})$ in the following way. The visual component, $|\mathcal{S}|_V$, of the carrier graph is defined to be the disjoint union of all $|\mathcal{D}_{F_k}|_V$, $k \in T$. The naming mapping $|\mathcal{S}|\colon |\mathcal{S}|_V \to Nms$ is obtained by composing naming mappings in shapes with diagrams in $T$-nodes, that is, by taking the coproduct of $|\mathcal{D}_{F_k}| \Rrightarrow \delta_k$, $k \in T$. $\Pi$-sketch structure on $|\mathcal{S}|$ is induced by graph morphisms $\delta_k$, $k \in T$: given $P \in \Pi$, $P$-diagrams in $\mathcal{S}$ are exactly those that are transferred from $\mathcal{D}_{F_k}$, $k \in T$. At last, $\Phi$-diagrams in $\mathcal{S}$ are directly given by $\delta_k$, $k \in T$.

In addition, a $\Pi$-subsketch of $\mathcal{S}$, $\mathcal{B}_\Pi = Ske_\Pi^{\mathrm{in}}(\mathcal{T})$, is designated by restricting the range of $k$ in all the items above by the set $\{k \in T \mid F_k \in Base\}$, *ie*, $\mathcal{B}_\Pi$ is the merge of all basic (without signature constants) leaf sketches. We make a default assumption that all $\Pi$-assertions associated with nodes of rank 1 are included into $\mathcal{B}_\Pi$ so that the rank of a node $k$ with $F_k \in Pred^+$ is always greater than 1. $\diamondsuit$

**3.2.14 Definition.** Let $\Sigma$ be a signature of markers as above.

(i) A *$\Sigma$-specification* is a pair $\mathbf{S} = (\mathcal{S}, \mathcal{B}_\Pi)$ where $\mathcal{S}$ is a $\Sigma$-sketch (that is, a $\Sigma_{pr}$-sketch) and $\mathcal{B}_\Pi$ is some distinguished $\Pi$-subsketch, $\mathcal{B}_\Pi \subset \mathcal{S}$. $\mathcal{S}$ is called the *carrying sketch* of $\mathbf{S}$ and $\mathcal{B}_\Pi$ is the *input sketch* of $\mathbf{S}$.

If to speak about visualization, then a $\Sigma$-specification is a visual $\Sigma$-sketch in which some *basic* subsketch is designated (by, say, painting items not occurring into it, *derived items*, with the green colour).

(ii) $\mathbf{S}$ is called *well-formed* (wf) if there is a well-formed $(\Sigma, Nms)$-tree $\mathcal{T}$ and an indexed $\Sigma$-sketch isomorphism $\pi\colon \mathcal{S}^\star \to [Ske_\Sigma(\mathcal{T})]^\star$ s.t. the restriction $\pi\big\lceil_{\mathcal{B}_\Pi^\star}$ is also an isomorphism $\mathcal{B}_\Pi{}^\star \cong [Ske_\Pi^{\mathrm{in}}(\mathcal{T})]^\star$. In particular, this means that $\mathcal{S} \simeq Ske_\Sigma(\mathcal{T})$ and $\mathcal{B}_\Pi \simeq Ske_\Pi^{\mathrm{in}}(\mathcal{T})$.

(iii) Well-formed $\Sigma$-specifications are also called *operational $\Sigma$-sketches*. $\diamondsuit$

Since our syntax is inductive the following is evident.

**3.2.15 Proposition (*Sketch parsing*).** There is an algorithm either extracting from any given specification $\mathbf{S} = (\mathcal{S}, \mathcal{B}_\Pi)$ a $(\Sigma, Nms)$-tree $\mathcal{T} = Tree(\mathbf{S})$ and isomorphism $Pars\colon \mathcal{S}^\star \to [Ske_\Sigma(\mathcal{T})]^\star$ as above, or terminated with failure proving that $\mathbf{S}$ is not well-formed.

Well-formed specification is called *free* if $Tree(\mathbf{S})$ is free. $\diamondsuit$

A simple example of sketch parsing is presented on Fig. 8.

**3.2.16 Proposition.** Let $\mathbf{S} = (\mathcal{S}, \mathcal{B}_\Pi)$ be an operational $\Sigma$-sketch (wf specification) and $e\colon \mathcal{B}_\Pi \to \mathcal{A}_\Pi$ be a $\Pi$-sketch-morphism into the $\Pi$-carrier of some $\Sigma$-algebra $\mathcal{A}$. Then there is no more than one expansion of $e$ to a $\Sigma$-sketch-morphism $\overline{e}\colon \mathcal{S} \to \mathcal{A}$ for which the restriction $\overline{e}\big\lceil_{\mathcal{B}_\Pi}$ equals $e$. In addition, if $\mathbf{S}$ is free then any $e\colon \mathcal{B}_\Pi \to \mathcal{A}_\Pi$ is expandable. $\diamondsuit$

**3.2.17** Given $\mathbf{S} = (\mathcal{B}_\Pi, \mathcal{S})$ and $e\colon \mathcal{B}_\Pi \to \mathcal{A}_\Pi$, the fact that $e$ is expandable means that all constraints (equations and $\Pi$-assertions) embodied into $\mathbf{S}$ are satisfied by $e$, we then write $e \models \mathbf{S}$ following the standard notation. If $\mathcal{A}{=}\mathcal{U}$ and $\mathbf{S}$ is understood as a semantic database schema, an expandable morphism $e\colon \mathcal{B}_\Pi \to \mathcal{A}_\Pi$ should be called a *state* (or *instance*) of the universe (database) if to follow the terminological tradition of the relational DB theory. Thus, a given schema $\mathbf{S} = (\mathcal{B}_\Pi, \mathcal{S})$ determines a set of states,

$$Stt(\mathbf{S}) \stackrel{\text{def}}{=} \{e\colon \mathcal{B}_\Pi \to \mathcal{U}_\Pi \mid e \models \mathbf{S}\}\,.$$

Unfortunately, a natural CT assumption that morphisms between states are ordinary natural transformations is not appropriate in the DB context: it was shown in [18] that the picture should be much more complex, and, in particular, involve world's evolution in order to capture object identities – a fundamental concept of software engineering ([30]). $\diamondsuit$

**3.2.18 Problem.** To find a proper definition of universe (database) state morphisms and study the corresponding category, $\mathbf{Stt}(\mathbf{S})$. Of course, there may be different definitions in function of context: there are different kinds of state transformations. $\diamondsuit$

Let $\mathcal{W}$ be a set of states of some semantic universe (world) $\mathcal{W}$. CT teaches us that it is natural to take $\mathcal{W}$ to be a topos (more accurately, computationally feasible, or *bounded* topos, cf.[36]). In the terms developed above, the principal task of semantic modeling is to find a semantic schema $\mathbf{S} = (\mathcal{B}_\Pi, \mathcal{S})$ and an adjunction (ideally, equivalence) between categories $\mathcal{W}$ and $\mathbf{Stt}(\mathbf{S})$ so that any world state would be representable in $\mathbf{Stt}(\mathbf{S})$ and, the reverse, any database state should represent some state of the world. Thus, the art of semantic modeling can be phrased in CT terms as an art of discovering effective and comprehensible presentations of bounded toposes. The present paper suggests to drain a little this heuristic area by adopting the sketch notation and thinking.

## 4 Towards graph-based finite model theory: Sketching metafinite structures

**4.1** A peculiarity of sketches appeared in database design is that their nodes are partitioned into abstract object classes and predefined value domains. Extension of the former consists of real world objects like persons or married couples while extension of the latter consists of values like integers or strings. So, if $\mathcal{S}$ is a semantic schema-sketch, its underlying graph contains a subgraph of value domains and (computable)

functions between them. In addition, the domain subgraph must be terminal in a certain sense (see [18] for details). This terminality reflects a principal property of the database technology: to be representable in the information system real world objects must be characterized by values processable by computer. Formal semantics of distinguishing between objects and values, and representing objects by values is a highly-nontrivial question, it was solved in [18] in the framework of variable sets semantics for sketches. Also, it was shown there that a principal component of semantic data specification is assignment to each class node a designated set of outgoing arrows, the *key* of the node. Correspondingly, correct data schemas should be *keyed* sketches and, moreover, they must be *well keyed* sketches: the key subgraph of a sketch must satisfy certain conditions to guarantee a *finitary* identification of objects by values. In particular, the terminality condition mentioned above is terminality of the set of domains in the key subgraph.

A slightly different treatment was used in [33] where value domains do not at all appear in carrying graphs of sketches. This is however inconvenient when one needs to compose a reference between classes with an attribute of the target class. $\diamondsuit$

**4.2**

The fundamental role of *ObjectClass-vs-ValueDomain* dichotomy in applications of model theory to computer science was stressed in [25] (in another terminology) where the dichotomy is captured in the notion of *metafinite structure*. Briefly, the latter is a triple $(\mathbf{C}, \mathbf{D}, W)$ with $\mathbf{C} = (C, K^{\mathrm{C}})$ a finite structure (in Tarski's sense) of some vocabulary $K$, $\mathbf{D} = (D, L^{\mathrm{D}})$ a (maybe, infinite) structure of another vocabulary $L$ and $W$ is a family of *weight* functions $w\colon C^m \to D$. The essence of metafinite model theory is in a special intended logic where $\mathbf{C}$ and $\mathbf{D}$ are accessed via different logical means.

The simple definition of metafinite structures above is apt for mathematical study of the phenomenon but is hardly suitable for real engineering applications where a universe normally consists of dozens of sets and functions. To specify the latter in a comprehensible form one should consider $\mathbf{C}$ and $\mathbf{D}$ as sketch mappings $[\![\ ]\!]_{\mathrm{C}}\colon \mathcal{S}_{\mathrm{C}} \to \mathcal{U}_{Obj}$ and $[\![\ ]\!]_{\mathrm{D}}\colon \mathcal{S}_{\mathrm{D}} \to \mathcal{U}_{Val}$ where sketches $\mathcal{S}_{\mathrm{C}}, \mathcal{S}_{\mathrm{D}}$ specify the class and value domain structures respectively. former sketch is arbitrary while the latter should be a finite limit sketch. The payment for engineering applicability is that structural aspects of specification become complex and a precise description is by now an open problem. We will try to outline a rough picture of what might be expected from a theory to be developed.

Sketches $\mathcal{S}_{\mathrm{C}}, \mathcal{S}_{\mathrm{D}}$ above are, in general, sketches in different signatures, $\Sigma_{\mathrm{C}}, \Sigma_{\mathrm{D}}$, somehow related by a span in the category of signatures. By taking push-out of the span, $\Sigma_{\mathrm{C}}$ and $\Sigma_{\mathrm{D}}$ can be merged into an integrated signature $\Sigma^{+}$. Correspondingly, sketches $\mathcal{S}_{\mathrm{C}}, \mathcal{S}_{\mathrm{D}}$ are transformed into $\Sigma^{+}$-sketches $\mathcal{S}^{+}_{\mathrm{C}}, \mathcal{S}^{+}_{\mathrm{D}}$. Now the crucial for database technology representation of objects by values is specified by a mapping $\tau\colon \mathcal{S}^{+}_{\mathrm{C}} \to \mathcal{S}^{+}_{\mathrm{D}}$ which, in the database terminology, assigns to each class node in $\mathcal{S}_{\mathrm{C}}$ its type in $\mathcal{S}_{\mathrm{D}}$, and to each marked diagram in $\mathcal{S}^{+}_{\mathrm{C}}$ – a computer procedure operating data values structured according to the shape of the diagram (and denoted in $\mathcal{S}^{+}_{\mathrm{D}}$ by the same marker). Thus, database schema specification is a triple $(\mathcal{S}_{\mathrm{C}}, \mathcal{S}_{\mathrm{D}}, \tau)$ and its model $[\![\ ]\!]$ is given by (i) a $\Sigma_{\mathrm{C}}$-sketch morphism $[\![\ ]\!]_{\mathrm{C}}\colon \mathcal{S}_{\mathrm{C}} \to \mathcal{U}_{Obj}$, (ii) a $\Sigma_{\mathrm{D}}$-sketch morphism $[\![\ ]\!]_{\mathrm{D}}\colon \mathcal{S}_{\mathrm{D}} \to \mathcal{U}_{Val}$ and (iii) the following (maybe, lax) commutative diagram:

$$\begin{tikzcd} \mathcal{S}^+_{\mathrm{C}} \arrow[r, "[\![\ ]\!]^+_{\mathrm{C}}"] \arrow[d, "\tau"'] & \mathcal{U}^+_{Obj} \arrow[d, "[\![\tau]\!]"] \\ \mathcal{S}^+_{\mathrm{D}} \arrow[r, "[\![\ ]\!]^+_{\mathrm{D}}"] & \mathcal{U}^+_{Val} \end{tikzcd}$$

$\diamondsuit$

**4.3 Problem.** To find a precise description of the impressionistic picture above. $\diamondsuit$

## 5 Conclusion

Category theory is well known as a high-level polymorphic framework suitable for specifying complex structures and formalisms arising in *theoretical studies* (such as metamathematics or theoretical computer science). Quite unexpectedly, however, it turned out that even elementary category theory notions can be valuable in *practice* of software engineering: [29, 14] are probably the first reports on practical influence of CT in software engineering, and in summer of 1994 Boris Cadish and the present author, being inspired by our success in applying sketches for practical database design, wrote a manifesto to claim our belief in the extremely promising perspectives of incorporating CT into DB area ([9] is the updated version). By now, the joint work of the entire "CT-for-DB" community shows that CT possesses a great potential for SE applications and hopefully can pretend on the role of basic universal specification language (which itself is a hot problem in the area, [21]).

Of course, as with any other mathematical discipline, there is a gap between CT and engineering applications which must be spanned to make CT really helpful. The present paper has hopefully demonstrated that the sketch machinery is extremely promising in building the bridge CT–for–>SE. On the other hand, the machinery developed under SE-influence seems to be interesting in the pure CT context. SE is thus appears as a source of fruitful categorical ideas, SE–for–>CT: a good bridge should serve well in both directions. Thus, an overall idea behind the paper can be captured by the following brief formula:

CT<–ekS-Ske–>SE .

## References

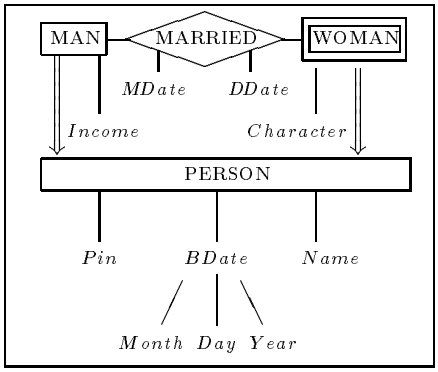

1. S. Abiteboul and A. Bonner. Objects and views. In *Proc.ACM SIGMOD Conf. on Management of Data*, pages 238–247, 1991.
2. J. Adámek, H. Herrlich, and J. Rosičky. Essentially equational categories. *Cahiers Topol. Géométr. Différ.*, 29(3):175–192, 1988.
3. R. Agrawal and L. G. DeMichiel. Type derivation using the projection operation. In *Int.Conf. EDBT'94*, Springer, LNCS'779, pages 7–14, 1994.
4. C. Batini, S. Ceri, and S. Navathe. *Conceptual database design: an Entity Relationship Approach.* Benjamin Cummings, 1992.
5. P. Buneman, S. Davidson, and A. Kosky. Theoretical aspects of schema merging. In *Proceedings of EDBT'92*, 1992.

6. A. Burroni. Algébres graphiques. *Cahiers de Topologie et Géométrie Diff.*, 22:249–266, 1981.
7. Cadish and Z. Diskin. Algebraic graph-oriented approach to view integration. Part I: Specification framework and general strategy. Technical Report 9301, Frame Inform Systems/LDBD, Riga, Latvia, 1993.
8. B. Cadish and Z. Diskin. Algebraic graph-based approach to management of multibase systems, I: Schema integration via sketches and equations. In *Next Generation of Information Technologies and Systems, NGITS'95*, 2nd Int.Workshop, pages 69–79, Naharia (Israel), 1995. (On ftp: //ftp.cs.chalmers.se /pub/users/diskin/ PAPERS-DB/ngits95.ps.gz).
9. B. Cadish and Z. Diskin. Algebraic Graph-Oriented = Category Theory Based. Manifesto of categorizing database theory. Technical Report 9506, Frame Inform Systems, 1995. (//ftp.cs.chalmers.se /pub/users/diskin/MANIFEST/mnfst4.ps).
10. B. Cadish and Z. Diskin. Heterogenious view integration via sketches and equations. In *Foundations of Intelligent Systems*, Proc. 9th Int.Symposium, *ISMIS'96*, Springer LNAI'1079, pages 603–612, 1996.
11. A. Chandra and D. Harel. Computable queries fo relational databases. *J.Computer System Sci.*, 21(2):156–178, 1980.
12. P.P. Chen. The entity-relationship model – Towards a unified view of data. *ACM Trans.Database Syst.*, 1(1):9–36, 1976.
13. C.N.G. Dampney, P. Deuble, and M. Johnson. Taming large complex information systems. In D.G. Green and T. Bossomaier, editors, *Complex Systems*. IOS Press, 1993.
14. C.N.G. Dampney, M. Johnson, and G.P. Monro. An illustrated mathematical foundation for ERA. In C.M. Rattray and R.G. Clarke, editors, *The Unified Computation Laboratory*. Oxford University Press, 1992.
15. Z. Diskin. Towards algebraic graph-based model theory for computer science. Abstract of talk accepted for "Logic Colloqium'95", to appear in "Bulletin of Symbolic Logic" (On ftp://ftp.cs.chalmers.se/pub/users/diskin/PAPERS-Math/lc95.*).
16. Z. Diskin and I. Beylin. What is an operation over sketches? A talk given at Winter Meeting of Computer Science Department, Chalmers University, Göteborg, January, 1995.
17. Z. Diskin and B. Cadish. Abstract queries, schema transformations and algebraic theories: An application of categorical algebra to database theory. In *Acta Universitatis Latviensis. Matematika*, volume 595, pages 83–96. University of Latvia, Riga, Latvia, 1994.
18. Z. Diskin and B. Kadish. Variable set semantics for generalized sketches: Why ER is more object-oriented than OO. To appear in *Data and Knowledge Engineering*, manuscript is available by ftp //ftp.cs.chalmers.se/pub/users/diskin/ER/ERvsOO.ps.
19. Z. Diskin and B. Kadish. Variable sets and functions framework for conceptual modeling: Integrating ER and OO via sketches with dynamic markers. In *OOER'95: Object-Oriented and Entity-Relationship Modeling*, Proc. 14th Int.Conf., Springer LNCS'1021, pages 226–237, 1995.
20. Z. Diskin and B. Kadish. Formalizing schemas for federal database environment architecture. Technical Report 9703, Frame Inform Systems, Riga, Latvia, 1997. (On ftp: //ftp.cs.chalmers.se /pub/users/diskin/REPORTS/tr9701.ps).
21. P. Drew, R. King, D. McLeod, M. Rusinkiewicz, and A. Silberschatz. Report on the workshop on semantic heterogeneity and interoperation in multidatabase systems. *SIGMOD Record*, 22(3):47–56, 1993.
22. E.J. Dubuc and G.M. Kelly. A presentation of topoi as algebraic relative to categories of graphs. *J. of Algbera*, 81:420–433, 1983.
23. H. Ehrig and B. Mahr. *Fundamentals of algebraic specifications I, Equations and initial semantics*. EATCS Monographs on Theoretical Computer Science, 6. Springer-Verlag, 1985.
24. P. Freyd and A. Scedrov. *Categories, Allegories*. North-Holland, 1992.

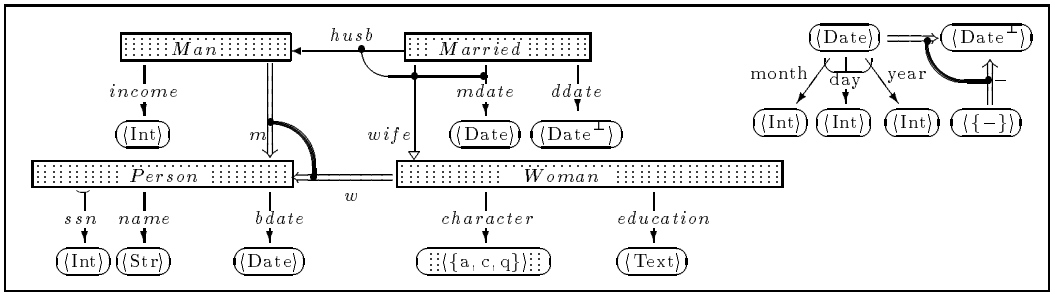
25. E. Gradel and Ju. Gurevich. Metafinite model theory. In D. Leivant, editor, *Logic and computational complexity*, pages 313–366. Springer, 1995.
26. J. Gray. The category of sketches as a model for algebraic semantics. In J. Gray and A. Scedrov, editors, *Categories in computer science and logic*, volume 92 of *Contemporary Mathematics*. American Mathematical Society, 1989.
27. T. Halpin. A fact oriented approach to schema transformation. In *3rd Symp.MFDBS'91*, Springer LNCS'495, pages 342–356, 1991.
28. A. Heuer and M. Scholl. Priciples of object-oriented query languages. In *Database systems for Office, Scientific and Engineering Applications*, pages 178–197, Kaiserslautern, 1991. Springer.
29. M. Johnson and C.N.G. Dampney. On the value of commutative diagrams in information modeling. In *Algebraic Methodology and Software Technology, AMAST'93*. Springer, 1993.
30. S.N. Khoshafian and G.P. Copeland. Object identity. In *ACM Conf. OOPSLA'86*, pages 406–415, 1986.
31. M. Makkai. Generalized sketches as a framework for completeness theorems. Technical report, McGill University, 1994. To appear in J.Pure Applied Algebra. (On ftp: //triples.math.mcgill.ca/pub/makkai/sketch).
32. S. Navathe. The next ten years of modeling, methodologies and tools. In *Entity Relationship modeling, ER'92*, number 645 in Springer LNCS'645, 1992.
33. F. Piessens and E. Steegmans. Canonical forms for data specifications. In *Computer Science Logic'94* 8th Int.Workshop, CSL'94, Springer LNCS'933, pages 397–411, 1994.
34. B. Plotkin. *Universal algebra, Algebraic logic and Databases*. Kluwer Publ.Co, 1993.
35. E. Robinson. Variations on algebra: monadicity and generalizations of equational theories. Technical report, University of Sussex, 1994. On ftp: //ftp/theory.doc.ic.ac.uk/papers/.
36. V. Yu. Sazonov. Hereditary finite sets, data bases and polynomial-time computability. *Theoretical Computer Science*, 119:187–214, 1993.
37. M.H. Scholl, H.-J. Schek, and M. Tresch. Object algebra and views for object bases. In *Int.Workshop on Distributed Object Management, Edmonton, Canada*, 1992.
38. S. Spaccapietra, C. Parent, and Y. Dupont. Model-independent assertions for integration of heterogeneous schemas. *Very Large Databases Journal*, 1(1), 1992.
39. S. Spaccapietra, C. Parent, and Y. Dupont. View integration: a step forward in solving structural conflicts. *IEEE Transactions on KDE*, 1992.
40. J. Van den Bussche, D. Van Gutch, M. Andries, and M. Gyssens. On the completeness of object-creating query languages. In *33rd Symposium on Foundations of Computer Science*, pages 372–379, 1992.
41. C. Wells and M. Barr. The formal description of data types using sketches. In *Mathematical foundations of Programming language semantics*, volume 298 of *Springer LNCS*, 1988.

This article was processed using the LaTeX macro package with LLNCS style

Semantic situation:
users are interested
in information
(say, for the last 50 years)
about men, married
couples and married women.

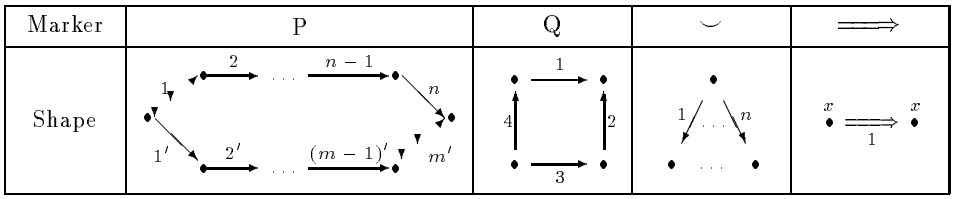



ER-diagram, $D$

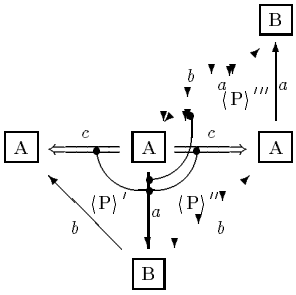



Sketch, $S$

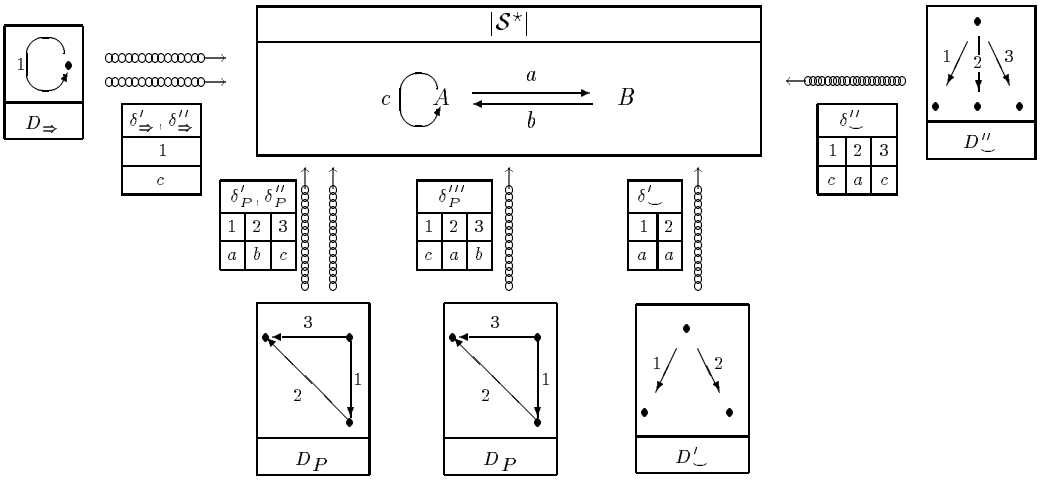
**Fig. 4.** Sketches vs ER-diagrams




a) Signature, $\Pi$




b) Visual $\Pi$-sketch, $\mathcal{S}$




c) Indexed sketch behind $\mathcal{S}$, $\mathcal{S}^\star$.
$\mathcal{S}^\star.P = \langle \delta'_P, \delta''_P, \delta'''_P \rangle$, $\mathcal{S}^\star.Q = \emptyset$, $\mathcal{S}^\star.\smile = \langle \delta'_\smile, \delta''_\smile \rangle$, $\mathcal{S}^\star.\Rightarrow = \langle \delta'_\Rightarrow, \delta''_\Rightarrow \rangle$

**Fig. 5.** Extraction of indexed sketch from a visual sketch

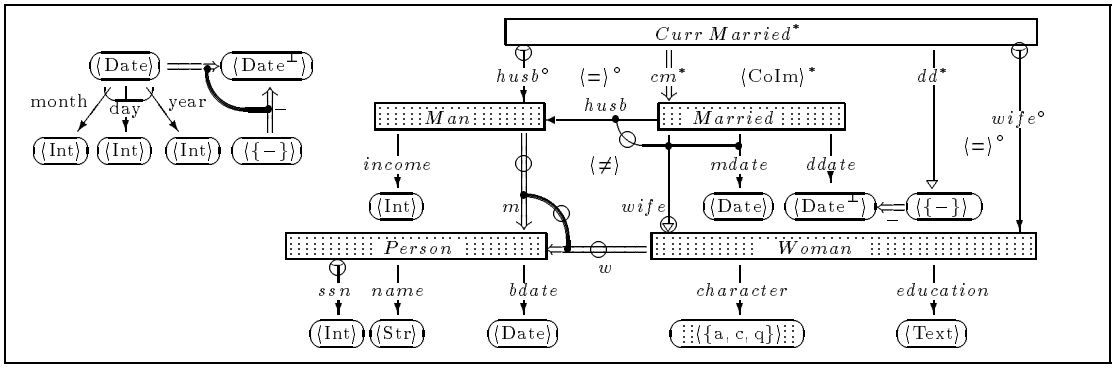



**Fig. 6.** Sketches with operation markers: Specifying a query to a sketch = Extending it with derived items

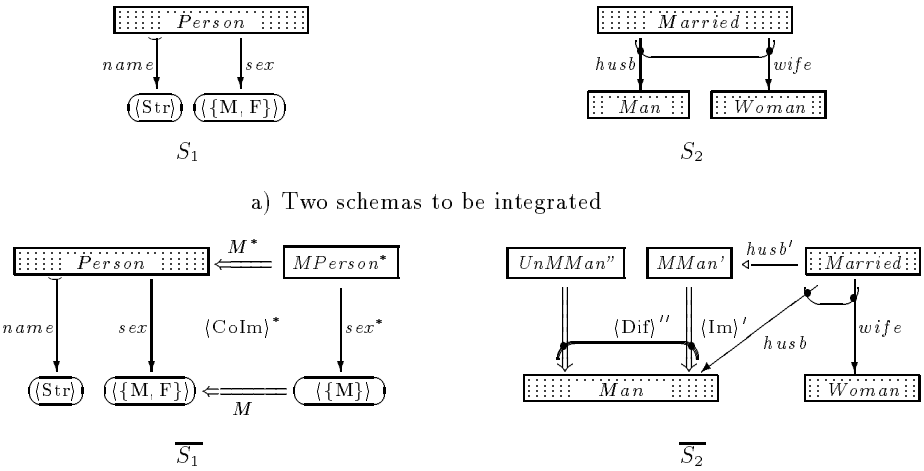
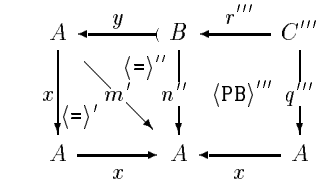
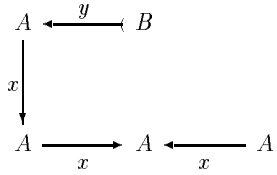


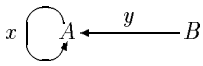

**Fig. 7.** Example of schema integration



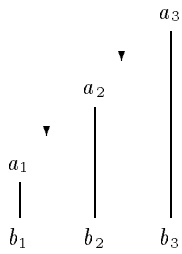
The carrying sketch of $\mathbf{S}, \mathcal{S}$



The input sketch of $\mathbf{S}, \mathcal{B}_{\Pi}$



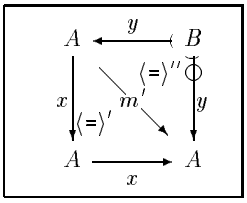
The name graph of $\mathcal{B}_{\Pi}$



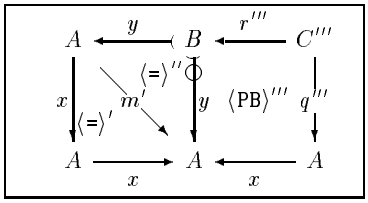
| Node | $b_1$ | $b_2$ | $b_3$ | $a_1$ | $a_2$ | $a_3$ |
|---|---|---|---|---|---|---|
| Marker | $C_{\downarrow\rightarrow}$ | $C_{\downarrow}$ | $C_{\downarrow}$ | $\langle = \rangle$ | $\langle = \rangle$ | $\langle \mathsf{PB} \rangle$ |
| Diagram | | | | | | |

The tree extracted from $\mathbf{S}$, $Tree(\mathbf{S})$

a) Free operational specification $\mathbf{S}{=}(\mathcal{S}, \mathcal{B}_{\Pi})$ and its parsing



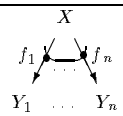
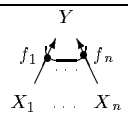
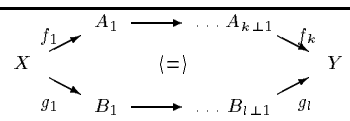
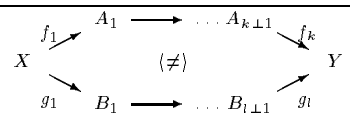
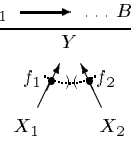
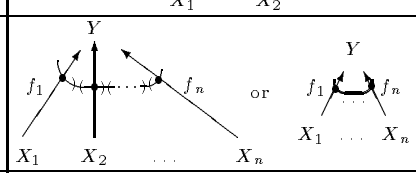
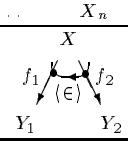
b) Assertional specification: $y \Rrightarrow (x \Rrightarrow x) = y$



c) Conditional operational specification:
if $y \Rrightarrow (x \Rrightarrow x) = y$
then $(C''', r''', q''') = PB(y, x)$

**Fig. 8.** Sketch parsing (shapes of markers )–, =, $PB$ are taken from Table 2 )

**Table 1.** A collection of diagram predicates

| Name | Arity Shape and Designation | Denotational Semantics |
|---|---|---|
| Separating Source |  | $(\forall x, x' \in X)\ x \neq x'$ implies $f_i(x) \neq f_i(x')$ for some $f_i$, i.e. there is an embedding $X \rightarrowtail Y_1 \times \cdots \times Y_n$ |
| Monic Arrow† | $X \overset{f}{\rightarrowtail} Y$ | $(\forall x, x' \in X)\ x \neq x'$ implies $f(x) \neq f(x')$ |
| Covering Flow |  | $(\forall y \in Y)(\exists i < n)\ y \in f_i(X_i)$ |
| Cover† | $X \overset{f}{\twoheadrightarrow} Y$ | $(\forall y \in Y)\ (\exists\, x \in X)\ y = fx$ |
| IsA-Arrow | $X \overset{f}{\Longrightarrow} Y$ | $X \subset Y$ and $f(x) = x$ for all $x \in X$ |
| Identity | $X \overset{f}{\Longleftrightarrow} X$ | $f(x) = x$ for all $x \in X$ |
| Commutativity |  | $(\forall x \in X)\ f_k \ldots f_1(x) = g_l \ldots g_1(x)$ |
| Anti-commutativity |  | $(\forall x \in X)\ f_k \ldots f_1(x) \neq g_l \ldots g_1(x)$ |
| Disjoint Images |  | $\{f_1 x : x \in X_1\} \cap \{f_2 x : x \in X_2\} = \emptyset$ |
| Disjoint Covering Flow |  | $(\forall y \in Y)(\exists i < n)\ y \in f_i(X_i)$ and $f_i(X_i) \cap f_j(X_j) = \emptyset$ for $i \neq j$ |
| $\epsilon$-relation |  | $(\forall y, y' \in Y_1)\ y \neq y'$ implies $\{f_2 x : x \in f_1^{\perp 1} y\} \neq \{f_2 x : x \in f_1^{\perp 1} y'\}$, i.e., there is an embedding $Y_1 \rightarrowtail \mathbf{Powerset} Y_2$ . |

†The construction is a trivial case of the previous one.

**Table 2.** A collection of diagram operations

| Name ⟨**marker**⟩ | Arity Shape: Input sketch | Arity Shape: Output sketch | Denotational Semantics | Linear notation |
|---|---|---|---|---|
| Identity ⟨**Id**⟩ | $\bullet$ $X$ | $X \xrightarrow{\langle \mathrm{id} \rangle} \bullet$ $f$ $X$ | $(\forall x \in X)\, fx = x$ | $f = id_X$ |
| Coimage† ⟨**CoIm**⟩ | $B \overset{b}{\Downarrow} X \xrightarrow{f} Y$ | $B' \xrightarrow{f'} B$, $b' \Downarrow$ ⟨CoIm⟩ $\Downarrow b$, $X \xrightarrow{f} Y$ | $B' = \{x \in X : fx \in B\}$<br>$f'$ = restriction of $f$ on $B'$ | $B' = f^{\perp 1}(Y)$<br>or else<br>$B' = CoIm_f(B)$ |
| Image† ⟨**Img**⟩ | $X \xrightarrow{f} Y$ | $I \overset{i}{\Longrightarrow} Y$, $f' \uparrow$ ⟨Img⟩ $\Vert$, $X \xrightarrow{f} Y$ | $I = \{f(x) : x \in X\}$<br>$(\forall x \in X)\, f'(x) = f(x)$ | $I = f(X)$,<br>or else<br>$I = Im_f(X)$ |
| Composition ⟨**=**⟩ | $Z \xrightarrow{g_2} Y$, $g_1 \uparrow$, $X$ | $Z \xrightarrow{g_2} Y$, $g_1 \uparrow$ ⟨=⟩ $\Vert$, $X \xrightarrow{f} Y$ | $(\forall x \in X) f(x) = g_2(g_1(x))$ | $f = g_1 \mathbin{\triangleright\!\!\triangleright} g_2$ |
| Graph ⟨**Gra**⟩ | $X \xrightarrow{f} Y$ | $G \xrightarrow{q} Y$, $p \downarrow$ ⟨Gra⟩ $\Vert$, $X \xrightarrow{f} Y$ | $G = \{(x, fx) : x \in X\}$<br>$p, q$ are projections | $G = Graph(f)$ |
| Pull-back† ⟨**PB**⟩ | $A \downarrow f$, $B \xrightarrow{g} X$ | $R \xrightarrow{g'} A$, $f' \downarrow$ ⟨PB⟩ $\downarrow f$, $B \xrightarrow{g} X$ | $R = \{(a, b) \in A \times B : fa = gb\}$<br>$p, q$ are projections | $R = PB(f, g)$ |
| Universal Arrow of PB† ⟨**!**⟩ | $Y \xrightarrow{e} A$, $h \downarrow$ ⟨=⟩, $R \xrightarrow{g'} A$, $f' \downarrow$ ⟨PB⟩ $\downarrow f$, $B \xrightarrow{g} C$ | $Y \xrightarrow{u} R$, $h \downarrow$ ⟨!⟩, $R \xrightarrow{g'} A$, $f' \downarrow$ ⟨PB⟩ $\downarrow f$, $B \xrightarrow{g} X$ | $(\forall y \in Y) uy = [ey, hy]$ | $u = UAPB(R, e, h)$ |
| Difference ⟨**Dif**⟩ | $A \Downarrow X$ | $A \Downarrow$ ⟨Dif⟩, $X \Leftarrow A'$ | $A' = \{x \in X : x \notin A\}$ | $A' = X \backslash A$ |
| Rule_1 | $A \leftarrow R$, $\Downarrow$, $X \Leftarrow B$ | $A \leftarrow R$, $\Downarrow$ ⟨≠⟩, $X \Leftarrow B$ | | |
| Rule_2 | $X \overset{f}{\Longrightarrow} Y$ | $X \overset{f}{\Longrightarrow} Y$ | | |

†Commutativity marker in the output diagram in suppressed