# REPORTS
# IN
# INFORMATICS

**ISSN 0333-3590**

**The Next Hundred**
**Diagrammatic Specification Techniques**
**— An Introduction to Generalized Sketches —**

**Uwe Wolter, Department of Informatics,**
**University of Bergen, Norway**

**Zinovy Diskin, Department of Computer Science, University of Toronto, Canada**

**REPORT NO 358** **July 2007**


*Department of Informatics*

**UNIVERSITY OF BERGEN**

*Bergen, Norway*

This report has URL http://www.ii.uib.no/publikasjoner/texrap/pdf/2007-358.pdf

Reports in Informatics from Department of Informatics, University of Bergen, Norway, is available at http://www.ii.uib.no/publikasjoner/texrap/.

Requests for paper copies of this report can be sent to:

Department of Informatics, University of Bergen, Høyteknologisenteret,
P.O. Box 7800, N-5020 Bergen, Norway

# Contents



**Abstract**

Generalized sketches is a graph-based specification format that borrows its main ideas from both categorical and first-order logic, and adapts them to software engineering needs. In the engineering jargon, it is a modeling language design pattern that combines mathematical rigor and appealing graphical appearance. The paper presents a revised framework of basic concepts to make similarities with the traditional FOL specifications transparent. [1] [2] [3]

# 1 Introduction and motivating discussion

Diagrammatic specifications are widely spread in software (and other branches of) engineering. Dozens of them were invented, raised, and forgotten, while many are still alive and prospering today. Amongst the latter are, for example, Peter Chen's entity-relationship (ER) diagrams, very popular in data modeling in the 70s, David Harel's statecharts, very popular in behavior modeling in the 80s, and message sequence charts (MSCs), very popular in telecom scenario modeling in the 90s. About ten years ago, these and some other diagrammatic notations were absorbed by an all-embracing industrial standard called UML and continue to dominate (either under the UML title or separately) the modeling segment of the industrial market and the academic market serving it.

Until recently, diagrammatic notations were mainly used as a communication medium between business experts, software designers and programmers. The goal was to communicate ideas and concepts, whose precision was (although very desirable) not a must. That explains the current situation with diagrammatic modeling, where semantic meaning of a construct is usually approximate and fuzzy (reaching sometimes a completely meaningless state, in which case the experts advise to consider it as a modeling placebo [RJB04]).

This is the state-of-the-art of the area, and there is a spectrum of reasons to be not entirely happy about it. At one end, there are arguments to have the meaning of notational constructs clear and precise for either facilitating communication or just for satisfying an understandable intellectual attitude, which makes using a technical langauge with unclear meaning quite uncomfortable. At the other end, there is a number of practical factors that have recently emerged in software industry, which we will not consider in detail. They are all subsumed under the titles of Model-Driven Engineering (MDE), or Model-Driven Development (MDD), or Model-Driven Architecture (MDA): different names of basically the same movement aimed at making models rather than code the primary artifacts of software development and at generating code directly from models [Sel06]. Needless to say that for MDD and friends, having a precise formal semantics for diagrammatic notations they employ is an absolute must. The industrial demand greatly energized building formal semantics for diagrammatic languages in use, and an overwhelming amount of them was proposed. The vast majority of them employ the familiar first-order (FO) or similar logical systems based on string-based formulas, and fail to do the job because of the following.

[1] Research partially supported by the Norwegian NFR project SHIP/VERDIKT.

[2] Supported by OCE Centre for Communications and Information Technology, IBM CAS Ottawa and IBM Eclipse Innovation Grant

[3] This paper is based on an invited talk at LSFA'06, September 17th 2006, Natal, Brazil and presents the material developed for a course on Generalized Sketches in fall 2006 at the University of Bergen.

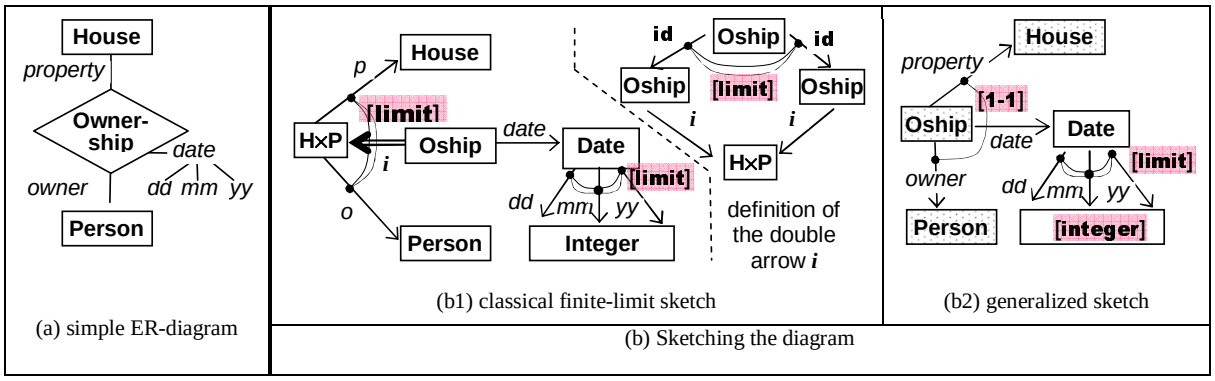



Figure 1: A sample of sketching a diagrammatic notation

Evidently, a necessary factor in a modeling language's success is its capability to capture the important aspects of the universe to be modeled and present them in as much direct and transparent as possible way.[4] A key feature of universes modeled in software engineering is their fundamental *conceptual two-dimensionality* (further referred to as 2D): entities and relationships (ERDs), objects and links (static, UML class diagrams, and dynamic, UML collaboration diagrams), states and transitions (statecharts), events and messages (MSCs), agents and interactions; the row can be prolonged.[5] Each of these conceptual arrangements is quite naturally represented by a graph – a 2D-structure of nodes and edges; the latter are usually directed and appear as arrows. In addition, these 2D-structures capture/model different aspects of the same whole system and hence are somehow interrelated between themselves. (For example, events happen to objects when the latter send and receive messages over dynamic links connecting them. These events trigger transitions over objects, which change their states). Thus, we come to another graph-based structure on the metalevel: nodes are graphs that model different aspects of the system and arrows are relations and interactions between them. The specificational system of aspects, views and refinements can be quite involved and results in a conceptually multi-dimensional structure. However complicated it may seem, this is the reality modern software engineers are dealing with (and languages like UML try to specify). Any attempt to describe this multidimensional universe in terms of FO or similar logics based on string-based formulas talking about elements of the domains rather than their relationships flattens the multi-level structure and hides the connections between the levels. This results in a bulky and unwieldy specifications, which are difficult (if at all possible) to understand, validate, and use.

A principally different approach to specifying structures, which focuses on relationships between domains rather then their internal contents and hence is essentially graph-based, was found in category theory (CT). It was originated by Charles Ehresmann in the 60s, who invented the so called *sketches* (see [Wel] for a survey); later sketches were promoted for applications in computer science by Michael Barr and Charles Wells [BW95] and applied to data modeling problems by Michael Johnson and Roberth Rosebrugh [JRW02]. The essence of the classical sketch approach to specifying data idea is demonstrated by Fig. 1.

Figure 1(a) shows a simple ER-diagram, whose meaning is hopefully clear from the names of its elements: we have a binary relation *Oship* over the sets *House* and *Person*, which also has an attribute *date*. In fact, the ER-diagram describes a configuration of sets and mappings, which in the classical Ehresmann's sketch framework is specified in Fig. 1(b1). The "limit" label is hung on the arrow span $(H \times P, p, o)$ (note the double arc) and declares the span to possess a special *limit* property specified in any CT-textbook and making the set $H \times P$ the Cartesian product of *House* and *Person* (see, e.g., [BW95] for details). Similarly, the set *Date* is declared to be the Cartesian cube of the set *Integer* of natural numbers. Finally, the separate limit diagram in the right upper corner forces the arrow $i$ to be injective by a standard categorical argument.

The limit predicate is only one amongst a family of the so called universal properties of sets-and-mappings diagrams found in CT. This family of predicates, though compact, is extremely expressive and allows us to express many properties of sets-and-mappings configurations appearing in practice, particularly, in semantic interpretations of ER-diagrams.[6] In this way we come to the sketch specification of ERD-semantics or, as we will say, *sketching*

[4] There are, of course, other crucial factors from cognitive and psychological to cultural and political, which are beyond the scope of this paper.

[5] We are indebted to Bran Selic for bringing the conceptual two-dimensionality metaphor onto the stage.

[6] In fact, since formal set theories can be encoded by universal predicates (by mapping them into toposes [LS86]), we can say that any

ER-diagrams. It provides a powerful mathematical framework for formalization and analysis of their semantics, where many useful results are obtained [JRW02, PS97]. The power of this framework is essentially based on a principal idea of algebraic, particularly, categorical, logic: semantics is a structure-preserving mapping (morphism) from a syntactical structure into a similarly structured semantic universe. Further we will refer to it as *Semantics-as-a-morphism (Saam) Principle*. [7]

Although mathematically elegant, the classical sketch approach has several inherent drawbacks in engineering applications. For instance, in our example, in order to declare a simple fact that *Oship* is a binary relation, we were forced to introduce a few auxiliary elements into our specification. Note also that while extensions of nodes *House* and *Person* are to be stored in the database implementing the specification, extension of node $H \times P$ is (fortunately!) not stored. On the other hand, some elements of the original diagrams (roles *property* and *owner*) are not immediately seen in the sketch and can only be derived (by compositions of $i$ with $p$ and $o$). Thus, before we assign a precise semantic meaning to ERD's elements, we need to apply to the diagram some non-trivial transformations, and only after that the Saam Principle can be used. From the view point of a software engineer, these transformations look artificial, unnecessary, and misleading.

Fortunately, the deficiency of the classical sketch framework mentioned above can be fixed without giving up the Saam Principle, and hence preserving all benefits that algebraic logic brings to the subject. The idea is demonstrated in Fig. 1(b2). We still want to specify the type of *Oship*-elements externally via mappings rather than internally (like it is done in FOL), but we do not want to introduce Cartesian product $H \times P$ into the specification. The crucial observation that allows us to do the job is well-known in CT: for a given span of mapping, e.g., $S = (\mathit{Oship}, \mathit{property}, \mathit{owner})$, its head *Oship* is isomorphic to a relation iff the leg mappings possess a special property of being *jointly injective* or *jointly [1-1]* (see below in example 5). Thus, we declare the span $S$ to be jointly [1-1] and come to the specification shown in Fig. 1(b2). Note that in this specification, [Integer] is not the name of the node but a predicate label declaring the extension of the node to be the set of integers. Thus, the specification in Fig. 3(b2) presents a graph, in which three diagrams are marked by predicate labels taken from a predefined signature. If $\Theta$ denotes the signature, we will call such specifications *(generalized)* $\Theta$*-sketches*.

Note that the sketch in Fig. 1(b2) is visually similar to the ER-diagram. We can even consider the very ER-diagrams as nothing but a visual representation of the sketch, in which the diamond node is just a special way to visualize the declaration of [1-1]-injectivity predicate. In this way the generalized sketches treatment (*sketching*) of ER-diagrams offers both (i) a precise formalization of their semantics and (ii) a framework that is visually appealing and transparent; as our experience shows, it can be readily understood by a software engineer. Sketching other notations used in software engineering can be approached in a similar way (see [Dis03] for a discussion). This is a part of the two ongoing projects in the University of Bergen and the Queen's University in Kingston, and the results obtained so far look very promising. They provide a base for an ambitious thesis that any diagrammatic notation in real practical use in software engineering is nothing but a specific visualization of the universal sketch specification pattern.

On the other hand, the sketch formalism appears to be a good design pattern for diagrammatic language design. It suggests that if one is thinking about designing a new language, one should answer, first of all, to the following three questions: what is the interpretation of nodes, what is about arrows, and what is the signature of diagram predicates that matter. The brevity, clear semantics and solid mathematical foundations can make this pattern really helpful in practice. Moreover, an active promotion of modeling techniques into software industry and, particularly, the rapid progress of the *domain-specific languages* sector (so called DSL), form an explicit industrial demand to convenient and handy yet reliable language design techniques. We believe that the generalized sketch pattern provides a promising theoretical supply to the demand. It would not be a big exaggeration to say that modeling languages are designed daily in the modern software industry. Hopefully, the next one hundred of modeling languages will be designed along the lines of the generalized sketch pattern.

Our goal in the paper is to present the very basic definitions of the theory in a well-structured way so that similarities with the traditional FOL Specification Framework would become transparent. We also consider dependance relations between (diagram) predicate symbols and explicitly model them by arrows between arity shapes. On one hand, this is a graph-based analog of (universally quantified) implications of FOL, which are very important for the latter. On the other hand, it makes the predicate signature a graph rather than a set and will allow us to present Makkai's multi-step procedure of building generalized sketches in a elegant way as a graph morphism. Throughout the paper, we essentially use the concept of a category but hopefully in such a way that readers not familiar with Category Theory [BW95, Fia05] can also grasp the content of the paper.

formalizable property of sets-and-mappings configurations can represented in the sketch language.

[7] In computer science this fundamental idea is known as denotational semantics.

We will only sporadically touch some details of applying sketches to formalizing semantics of ER- and UML-class diagrams; the interested reader should consult [PS97, JRW02, DK03]. We omit also the important issue of "good" (suggestive and user-friendly) visualization of sketches including the discussion of what is a good visualization of a formal specification (some preliminary discussion can be found in [Gog98, Dis02, FB05]).

# 2 Graphs and Diagrams

The Generalized Sketch framework, as any other diagrammatic specification technique, is heavily based on the concept of graph, thus we start with a formal definition of this concept and the other necessary concepts around. Before we dive into technicalities, the following important remark is in order. In this paper we are concerned with (at least) three different kinds of "diagrams". To minimize potential confusion, we will try to distinguish between them with the following terminology.

1. There is the general idea of a "picture" with (labeled) nodes and (labeled) edges. We will refer to those pictures by the term **pict-diagrams**.
2. We have different specializations of this general idea for specification purposes like "ER diagrams", "UML class diagrams", . . .. We will refer to those specializations as **spec-diagrams**.
3. Finally, we have a strict, formal mathematical definition of diagrams. We will call these diagrams **math-diagrams** or simple **diagrams**.

We adapt the notation from [Fia05].

**Definition 1** (Graph)**.** *A **graph** $G = (G_0, G_1, sc, tg)$ is given by*

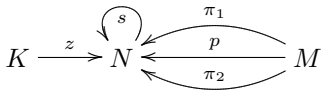
- *a collection $G_0$ of nodes,*
- *a collection $G_1$ of arrows,*
- *a map $sc : G_1 \to G_0$ assigning to each arrow its **source**, and*
- *a map $tg : G_1 \to G_0$ assigning to each arrow its **target**.*

*We usually write $f : x \to y$ or $x \xrightarrow{f} y$ to indicate that $sc(f) = x$ and $tg(f) = y$.*

*A graph $G = (G_0, G_1, sc^G, tg^G)$ is **subgraph** of a graph $H = (H_0, H_1, sc^H, tg^H)$, $G \sqsubseteq H$ in symbols, iff $G_0 \subseteq H_0$, $G_1 \subseteq H_1$, and $sc^G(f) = sc^H(f)$, $tg^G(f) = tg^H(f)$ for all $f \in G_1$.* □

**Remark 1** (Graphs)**.** *Graphs as defined in Definition 1 can be seen as many-sorted algebras with two sorts and two unary operations. This algebraic concept of graph allows for "muliple arrows". That is, between two nodes there may exist no arrows, just one in either direction, or several arrows, possibly in both directions. Note, that in case there is at most one arrow between any two nodes an arrow is uniquely determined by its source and target, i.e., we can assume, in this case, $G_1 \subseteq G_0 \times G_0$ where $sc$ and $tg$ are given by the corresponding projections, respectively.* □

**Example 1** (Finite Graphs)**.** *Finite graphs are often visualized by pict-diagrams. The graph $G$ with $G_0 = \{K, N, M\}$, $G_1 = \{z, s, p, \pi_1, \pi_2\}$ and $sc(z) = K$, $tg(z) = sc(s) = tg(s) = N$, $sc(\pi_1) = sc(\pi_2) = sc(p) = M$, and $tg(\pi_1) = tg(\pi_2) = tg(p) = N$, for example, can be visualized by the pict-diagram*




**Example 2** ("Big" Graphs)**.** *Beside finite graphs we consider also "very big" graphs as, e.g., the graph* Set *with nodes all sets $A$ and arrows all maps $f : A \to B$. Later, we will make use of the fact that* Set *is a **category**, i.e., for any set $A$ there is an **identical map** $id_A : A \to A$ and for any two maps $f : A \to B$, $g : B \to C$ there is the **composition** $f; g : A \to C$ defined by $f; g(a) = g(f(a))$ for all $a \in A$. Composition is associative and the identical maps are neutral w.r.t. composition.*

*More flexibility for defining semantics of Diagrammatic Techniques like ER diagrams is provided by the category* Par *with nodes all sets $A$ and with arrows all **partial maps** $f : A \nrightarrow B$ where we denote the **domain of***

***definition*** *of $f$ by $dom(f) \subseteq A$. The identities $id_A : A \to A$ are the total identical maps and the composition $f;g : A \nrightarrow C$ of two of two partial maps $f : A \nrightarrow B$ and $g : B \nrightarrow C$ is defined by*

$$dom(f;g) \stackrel{def}{=} \{a \in A \mid a \in dom(f), f(a) \in dom(g)\} \quad \textit{and} \quad f;g(a) \stackrel{def}{=} g(f(a)) \textit{ for all } a \in dom(f;g)$$

*Another useful example is sketching the semantics of UML class diagrams [Dis03]. It is given by the category* Pow *with nodes all sets $A$ and with arrows all maps $f : A \to \mathcal{P}(B)$ where $\mathcal{P}(B)$ is the* ***power set*** *of $B$, i.e., the set of all subsets of $B$. The identities $id_A : A \to \mathcal{P}(A)$ are defined by $id_A(a) = \{a\}$ for all $a \in A$ and the composition $f;g : A \to \mathcal{P}(C)$ of two maps $f : A \to \mathcal{P}(B)$ and $g : B \to \mathcal{P}(C)$ is defined by*

$$f;g(a) = g(f(a)) \stackrel{def}{=} \bigcup \{g(b) \mid b \in f(a)\} \quad \textit{for all } a \in A$$

*By an abuse of notation we will use the same notation for a category and for its underlying graph.* □

**Definition 2** (Graph Homomorphism)**.** *A* ***graph homomorphims*** *$\varphi : G \to H$ is a pair of maps $\varphi_0 : G_0 \to H_0$ and $\varphi_1 : G_1 \to H_1$ such that for each arrow $f : x \to y$ of G we have $\varphi_1(f) : \varphi_0(x) \to \varphi_0(y)$ in H, i.e., we have $sr^H(\varphi_1(f)) = \varphi_0(sr^G(f))$ and $tg^H(\varphi_1(f)) = \varphi_0(tg^G(f))$ for all $f \in G_1$.*

*The* ***composition*** *$\varphi;\psi : G \to K$ of two graph homomorphisms $\varphi : G \to H$ and $\psi : H \to K$ is defined component-wise*

$$\varphi;\psi = (\varphi_0, \varphi_1);(\psi_0, \psi_1) \stackrel{def}{=} (\varphi_0;\psi_0, \varphi_1;\psi_1).$$

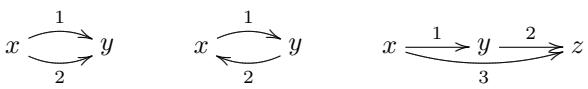
**Remark 2** (Category of Graphs)**.** *It is immediate to see that the composition $\varphi;\psi : G \to K$ is indeed a graph homomorphism. Identical graph homomorphisms $id_G : G \to G$ are defined by $id_G = (id_{G_0}, id_{G_1})$, and the component-wise definition of identities and composition ensures that we inherit associativity and neutrality from the category* Set*. In such a way, we obtain the category* Graph *of graphs and graph homomorphisms.*

*Note, that $G \sqsubseteq H$ iff the inclusion maps $in_i : G_i \hookrightarrow H_i$, $i = 0, 1$ define a graph homomorphism $in = (in_0, in_1) : G \hookrightarrow H$.*

A first important **methodological point** for Diagrammatic Specification Techniques is that we have to distinguish clearly between the concept of a graph and the concept of a diagram.

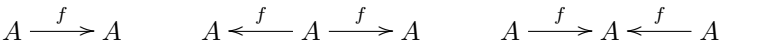
**Definition 3** (Diagram)**.** *Let $G$ and $I$ be graphs, A* ***diagram*** *in $G$ with* ***shape*** *$I$ is a graph homomorphism $\delta : I \to G$.*

**Example 3** (Shapes)**.** *Some of the most simple shapes, that are used in nearly any application, are* `Node` $= (\ x\ )$, `Arrow` $= (x \stackrel{1}{\to} y)$, `Span` $= (x \stackrel{1}{\leftarrow} z \stackrel{2}{\to} y)$, `Cospan` $= (x \stackrel{1}{\to} z \stackrel{2}{\leftarrow} y)$*, and the following three graphs* `Cell`*,* `Circle`*, and* `Triangle`*, respectively:*



**Remark 3** (Parameter)**.** *In programming we have the concepts "formal parameter list" and "actual parameter list". Analogously, a shape can be seen as a "formal parameter graph" with variables for nodes (as in a "formal parameter list") and, in generalizing the concept of list, also with variables for arrows. A diagram assigns "actual values" to the variables and can bee seen, in such a way, as an "actual parameter graph". A crucial point is that we can assign the same "actual value" to different variables thus in the "actual parameter graph" there maybe different copies of the same "actual value" (see also Remark 4).*

**Remark 4** (Visualization of Diagrams)**.** *Pict-diagrams are traditionally also used to present diagrams $\delta : I \to G$. The essential idea is to draw a picture with a separate "place holder node" for each element in $I_0$ and with a separate "place holder arrow" for each element in $I_1$. And then we put into the "place holders" the corresponding items from $G$ according to the assignment $\delta$. In such a way, we may have in the picture different copies of the same item from $G$ at different places. The following three pict-diagrams, for example,*



*represent diagrams $\delta_1$ :* `Arrow` $\to$ Set*, $\delta_2$ :* `Span` $\to$ Set*, $\delta_3$ :* `Cospan` $\to$ Set *with the same "actual values" but with different shapes.*

*A critical **methodological point** is, that the ordinary way of visualizing math-diagrams by pict-diagrams can be ambiguous. From the following pict-diagram*

$$N \xleftarrow{\pi_2} M \xrightarrow{\pi_1} N$$

*visualizing a diagram $\delta : \mathtt{Span} \to G$ in the graph $G$ from Example 1, we cannot deduce, for example, the following information: does $\delta$ actually assign to $1$ the arrow $\pi_1$ or the arrow $\pi_2$? This information may be irrelevant in this special case, but it illustrates that some additional information may be required to extract the respective math-diagram from a given pict-diagram, and a tool for drawing diagrams must handle those informations.*

# 3 Generalized Sketches

In this section we present the concept of Generalized Sketches. Any (generalized) sketch $\mathcal{S}$ has an underlying "diagram signature" $\Theta$. And the overall idea/claim concerning the potential rôle of the Generalized Sketch framework within the area of formal specification could be formulated as follows: For any (diagrammatic) specification formalism **T** we can find a "diagram signature" $\Theta_{\mathbf{T}}$ such that for each **T**-specification (**T**-diagram) $D$ there is at least one $\Theta_{\mathbf{T}}$-sketch $\mathcal{S}_D$ such that $D$ can be seen as a (visual) presentation of $\mathcal{S}_D$. Thereby $D$ appears to be ambiguous if there are different $\Theta_{\mathbf{T}}$-sketches represented by $D$. That is, in some cases the transition from $\Theta_{\mathbf{T}}$-sketches to **T**-specifications (**T**-diagrams) involves a loss of information. We will call the creative process of finding an appropriate "diagram signature" $\Theta_{\mathbf{T}}$ for a specification formalism **T** the process of **"sketching the formalism T"**.

**Definition 4** (Diagram Signature)**.** *A **(diagram) signature** $\Theta = (\Pi, \alpha)$ is given by*

- *a collection $\Pi$ of **(predicate) labels or symbols** and*
- *a function $\alpha : \Pi \to \mathsf{Graph}_0$ assigning to each label $P \in \Pi$ its **arity (shape)** $\alpha(P)$.*

**Remark 5** (Compound Labels)**.** *In many applications it may be more "user friendly" to allow to assign to a predicate label $P$ a set of possible arity shapes. We could, for example, have a label $[prod]$ with different shapes for empty, binary, ternary, ... products, respectively. We decided **not** to allow for those sets of arity shapes in the actual definition of signatures. This will provide us, for example, with more flexibility for defining signature morphisms.*

*In examples and applications a "user" is, of course, free to define a "generic predicate label" $P$ with a corresponding set of arity shapes. But, this will be interpreted as a "user defined mechanism" to create **compound labels**. In case of the above mentioned generic label $[prod]$ we could use, for example, the compound labels $([prod], 0)$, $([prod], 2)$, $([prod], 3)$, .... Such a "user defined mechanism" to create compound labels will be also helpful, for example, to create all the necessary labels with the same shape* `Arrow` *to reflect the potentially infinite many different cardinality constraints for associations in UML class diagrams (see Example 8).*

Specifications within the Generalized Sketch framework are meant to be

**Definition 5** ((Generalized) Sketches)**.** *Given a diagram signature $\Theta = (\Pi, \alpha)$ a **$\Theta$-sketch** $\mathcal{S} = (G^{\mathcal{S}}, \mathcal{S}(\Pi))$ consists of a graph $G^{\mathcal{S}}$ and a $\Pi$-indexed family $\mathcal{S}(\Pi) = (\mathcal{S}(P) \mid P \in \Pi)$ of sets of **marked diagrams**, i.e., for every label $P \in \Pi$ there is a (maybe, empty) set $\mathcal{S}(P)$ of diagrams $(P, \delta : \alpha(P) \to G^{\mathcal{S}})$ in $G^{\mathcal{S}}$ **marked by** $P$.*

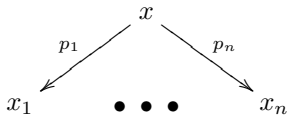
*A $\Theta$-sketch $\mathcal{S} = (G^{\mathcal{S}}, \mathcal{S}(\Pi))$ is **$\Theta$-subsketch** of a $\Theta$-sketch $\mathcal{T} = (G^{\mathcal{T}}, \mathcal{T}(\Pi))$, $\mathcal{S} \sqsubseteq \mathcal{T}$ in symbols, iff $G^{\mathcal{S}} \sqsubseteq G^{\mathcal{T}}$ and $\mathcal{S}(P) \subseteq \mathcal{T}(P)$ for all $P \in \Pi$.*

Now we want to look at how some basic traditional specification techniques can be reflected by diagram signatures and sketches.

**Example 4** (Algebraic Signatures)**.** *Algebraic signatures $\Sigma = (S, OP)$ declare a set $S$ of sort symbols and a set $OP$ of operation symbols together with corresponding arity requirements $op : s_1 \dots s_n \to s$. The only "semantic" requirement in this "specification formalism" is that for any $\Sigma$-algebra $\mathcal{A}$ the sequence $s_1 \dots s_n$ of sort symbols has to be interpreted by the cartesian product $\mathcal{A}(s_1) \times \dots \times \mathcal{A}(s_n)$ of the interpretations of the corresponding single sort symbols. Therefore, the sketching of the formalism "algebraic signatures" may lead us to a diagram signature $\Theta_{\mathbf{AS}} = (\Pi_{\mathbf{AS}}, \alpha_{\mathbf{AS}})$ that defines infinite many labels*

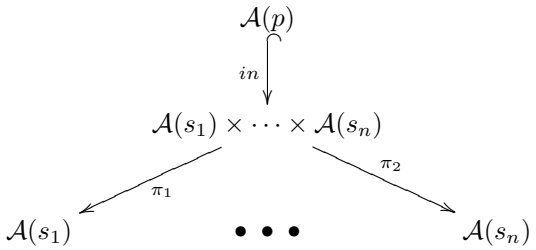
$$\Pi_{\mathbf{AS}} = \{([prod], 0), ([prod], 2), ([prod], 3), \dots, ([prod], n), \dots\}$$

*with arities* $\alpha_{\mathbf{AS}}([prod],0) = \texttt{Node}$, $\alpha_{\mathbf{AS}}([prod],2) = \texttt{Span}$ *and for any* $n \in \mathbb{N}$ *with* $n > 2$ *the arity* $\alpha_{\mathbf{AS}}([prod],n)$ *will be given by the following graph* $\texttt{Span}_n$




*A* $\Theta_{\mathbf{AS}}$*-sketch* $\mathcal{S}_{\Sigma_{Nat}}$ *that sketches the algebraic signature* $\Sigma_{Nat}$ *with* $S_{Nat} = \{N\}$ *and* $OP_{Nat} = \{z : \to N, s : N \to N, p : N\,N \to N\}$*, for example, is given by the graph* $G$ *in Example 1 together with the two marked diagrams* $\mathcal{S}_{\Sigma_{Nat}}([prod],0) = \{K\}$, $\mathcal{S}_{\Sigma_{Nat}}([prod],2) = \{N \xleftarrow{\pi_1} M \xrightarrow{\pi_2} N\}$*. All the other sets* $\mathcal{S}_{\Sigma_{Nat}}([prod],n)$ *will be empty for* $n > 2$*. Note, how the implicit assumption in* $\Sigma_{Nat}$ *that the sequence* $N\,N$ *of sort symbols has to be interpreted by a product has been made explicit in the sketch* $\mathcal{S}_{\Sigma_{Nat}}$*.*

**Example 5** (First-Order Signatures)**.** *First-order signatures* $\Sigma = (S, OP, P)$ *declare additionally to sort and operation symbols also a set* $P$ *of predicate symbols* $p$ *together with corresponding arity requirements* $p : s_1 \dots s_n$*. Thereby, for any* $\Sigma$*-model* $\mathcal{A}$ *the predicate* $\mathcal{A}(p)$ *is assumed to be a subset of the cartesian product* $\mathcal{A}(s_1) \times \dots \times \mathcal{A}(s_n)$*. We could introduce now, additionally to the labels* $([prod],n)$ *a predicate label* $[inj]$ *with arity shape* $\texttt{Arrow}$ *for marking a map as injective (mono). That is, we could start to define a traditional "limit sketch" [BW95] where we would have to code the property "mono" by a "pullback requirement". But, this would mean, that we have to introduce in the corresponding sketch* $\mathcal{S}_\Sigma$ *for each symbol* $p$ *an* ***auxiliary*** *node* $s_1 \dots s_n$ *as well as an auxiliary pullback diagram, in order to reflect the "semantic picture"*



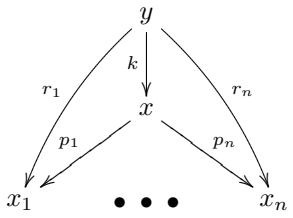

*Moreover, the projections* $\pi_i : \mathcal{A}(s_1) \times \dots \times \mathcal{A}(s_n) \to \mathcal{A}(s_i)$ *are not of primary interest, but the composed projections* $in; \pi_i : \mathcal{A}(p) \to \mathcal{A}(s_i)$*, and they are still not present in the picture.*

*It is more appropriated to introduce infinite many labels*

$$\Pi_{\mathbf{FS}} = \Pi_{\mathbf{AS}} \cup \{([1{-}1],1),([1{-}1],2),\dots,([1{-}1],n),\dots\}$$

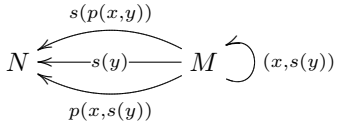
*with arities* $\texttt{Arrow}$, $\texttt{Span}$, ..., $\texttt{Span}_n$,..., *to indicate that the corresponding multiple spans of maps, are* ***"jointly injective (mono)"****. Two maps* $p_A : R \to A$, $p_B : R \to B$*, for example, are jointly injective iff* $p_A(r_1) = p_A(r_2)$ *and* $p_B(r_1) = p_B(r_2)$ *implies* $r_1 = r_2$ *for all* $r_1, r_2 \in R$*.*

**Example 6** (Equational Specifications)**.** *An* ***equational specification*** $SP = (\Sigma, EQ)$ *is given by an algebraic signature* $\Sigma = (S, OP)$ *and a set* $EQ$ *of* $\Sigma$*-equations* $(t_1 = t_2)$ *with* $t_1$ *and* $t_2$ $\Sigma$*-terms of the same sort. To reflect the concept of terms the intended diagram signature* $\Theta_{\mathbf{EQ}}$ *has to include, besides the "product labels"* $([prod],n)$ *from* $\Pi_{\mathbf{AS}}$*, additional labels: A reasonable choice could be a label* $[comp]$ *with arity* $\texttt{Triangle}$ *to reflect the compostion of maps, a label* $[=]$ *with arity* $\texttt{Cell}$ *to write equations, and compound labels* $([tupl],n)$*,* $n \geq 2$ *with arities given by the following diagrams* $\texttt{Tupl}_n$

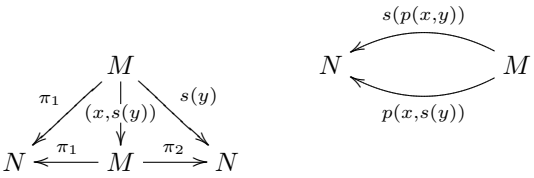



*to reflect the tupling of terms. To describe equations with a multiple occurence of a single variable we introduce further a label* $[id]$ *with arity* $\texttt{Arrow}$ *for indicating identical maps.*

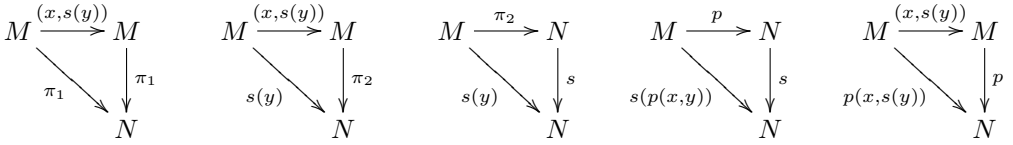
*A $\Theta_{\mathbf{EQ}}$-sketch $\mathcal{S}_{SP_{Nat}}$ that correponds to the equational specification $SP_{Nat} = (\Sigma_{Nat}, EQ_{Nat})$ with a single equation $EQ_{Nat} = \{(p(x, s(y)) = s(p(x, y))\}$, for example, will be given by the graph G in Example 1 extended by the following four arrows*



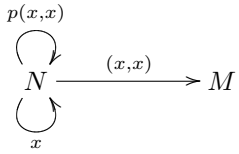

*There are no additional diagrams marked by product labels, i.e., we have $\mathcal{S}_{SP_{Nat}}([prod], n) = \mathcal{S}_{\Sigma_{Nat}}([prod], n)$, for $n = 0, 2, 3, \ldots$. The sets $\mathcal{S}_{SP_{Nat}}([tupl], 2)$ and $\mathcal{S}_{SP_{Nat}}([=])$ are singleton sets visualized by the following pict-diagrams*



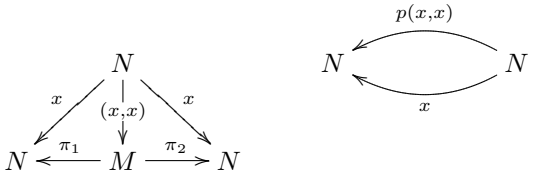

*$\mathcal{S}_{SP_{Nat}}([comp])$ contains five diagrams. The first two diagrams arise from tupling and the other three from composition.*



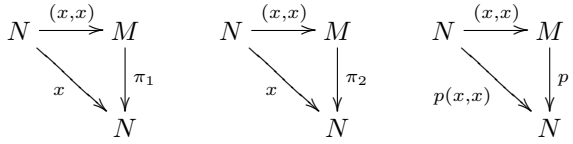

*Another $\Theta_{\mathbf{EQ}}$-sketch $\mathcal{S}_{SP'_{Nat}}$ that correponds to the equational specification $SP'_{Nat} = (\Sigma_{Nat}, EQ'_{Nat})$ with a single equation $EQ'_{Nat} = \{(x = p(x, x))\}$ is given by the graph G in Example 1 extended by the following three arrows*




*There are no additional diagrams marked by product labels, i.e., we have $\mathcal{S}_{SP_{Nat}}([prod], n) = \mathcal{S}_{\Sigma_{Nat}}([prod], n)$, for $n = 0, 2, 3, \ldots$. The sets $\mathcal{S}_{SP_{Nat}}([tupl], 2)$ and $\mathcal{S}_{SP_{Nat}}([=])$ are singleton sets visualized by the following pict-diagrams*




*$\mathcal{S}_{SP_{Nat}}([comp])$ contains three diagrams. The first two diagrams arise from tupling and the third from composition.*



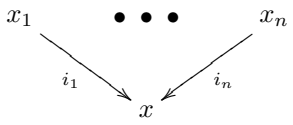

*Note, that the $\Theta_{\mathbf{EQ}}$-sketches $\mathcal{S}_{SP_{Nat}}$ and $\mathcal{S}_{SP'_{Nat}}$ have to present explicitly all the subterms in the equations $(p(x, s(y)) = s(p(x, y))$ and $(x = p(x, x))$, respectively. In traditional equational specifications this is not necessary since the syntactical appearance of a term allows to reconstruct the inductive process of constructing the term. This feature can be seen, somehow, as the essence of the concept of term. At the present stage the Generalized Sketch framework has not incorporated such advanced possibilities to create syntactical denotations (see also the discussion in Remark 11).*

**Example 7** (ER diagrams)**.** *Sketching ER diagrams with generalized sketches is discussed in detail in [DK03]. Here we want to discuss only some basic ideas and observations.*

*Firstly, it is appropriate to have in* $\Theta_{\mathbf{ER}}$ *predicate labels* $[entity]$, $[rel]$, $[attr]$ *with arity* `Node` *to distinguish between nodes that stand for "entities", "relation ships", or "attribute values", respectively. Since, it is usually allowed to have partially defined attributes the category* `Par` *of partial maps would be the appropriate choice to formalize the (static) semantics of ER diagrams. This means, that* $\Theta_{\mathbf{ER}}$ *should contain a label* $[total]$ *with arity* `Arrow` *to mark the total maps.*

*The actual concept of a "relation ship" can be reflected by* $([prod], n)$*-marked diagrams with arity* $\mathtt{Span}_n$*,* $n \geq 2$*. The arrows in* $\mathtt{Span}_n$ *should be assigned to* $[total]$*-marked arrows only, and the node* $x$ *in* $\mathtt{Span}_n$ *should be assigned to* $[rel]$*-marked nodes only (see the discussion of dependencies in Section 6). We will not allow for "aggregations", if we require additionally that the nodes* $x_i$ *are assigned only to* $[entity]$*-marked nodes.*

*To reflect the possibility that an entity can be the (disjoint) union of two or more other entities, it is appropriate to have generic labels* $([cover], n)$ *and* $([disj], n)$ *with arity* $\mathtt{Cospan}_n$*,* $n \geq 2$*,*




*where* $([cover], n)$ *indicates that the involved* $n$ *maps with the same target are "jointly surjective". A label* $([cover], 1)$ *with arity* `Arrow` *indicates that the corresponding arrow is surjective.*

*Note, that in ER diagrams the property "surjective" is usually visualized not by labeling the actual arrow but the target node instead. And, the property "jointly surjective" is visualized not by labeling the family of arrows as a unity but by labeling the single arrows instead. This kind of "immature visualization" cause some of the ambiguities in ER diagrams.*

**Example 8** (Associations)**.** *Here we want to discuss shortly a simplified version of UML* ***associations***.[8] *We restrict ourselves to a static viewpoint and consider semantics of a class to be a set. An association*


*describes then a relationship between two sets* $A$ *and* $B$*. But, in contrast to first-order signatures and ER diagrams, this relationship is thought of as a pair of multivalued functions (called* association ends *in UML),* $f : A \to \mathcal{P}(B)$ *and* $g : B \to \mathcal{P}(A)$ *rather than as a subset* $R \subseteq A \times B$ *of the Cartesian product. However, both functions have the same extension* $R \subseteq A \times B$ *so that* $f(a) = \{b \in B \mid (a, b) \in R\}$ *for all* $a \in A$ *and* $g(a) = \{a \in A \mid (a, b) \in R\}$ *for all* $b \in B$*. In other words, the two functions are* ***inverse*** *to each other in the sense of the following equivalence*

$$b \in f(a) \Leftrightarrow a \in g(b) \Leftrightarrow (a, b) \in R \quad \textit{for all} \quad a \in A,\ b \in B.$$

*To specify this fact in the generalized sketch framework, we should introduce into* $\Theta_{\mathbf{UML}}$ *a label* $[inv]$ *with arity* `Circle`*. Note, that in a concrete UML class diagram there may be only one of the two ends to restrict access and navigation.*

*UML offers a great freedom to restrict the cardinality of the association ends. The following diagram on the left*

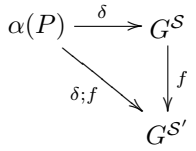

*expresses the restriction* $0 \leq |f(a)| \leq 1$ *for all* $a \in A$*, i.e.,* $f$ *is "single valued" and can be seen as a usual partial map. The restriction* $1..*$ *on* $f$ *in the middle diagram above means* $1 \leq |f(a)|$ *for all* $a \in A$*, i.e.,* $f$ *is "total". Note that this restriction on* $f$ *entails that* $g$ *is "covering", i.e. surjective, in the sense that* $\bigcup\{g(b) \mid b \in B\} = A$*, since* $f$ *and* $g$ *are inverse to each other. We could now introduce into* $\Theta_{\mathbf{UML}}$ *labels* $[single]$*,* $[total]$*,* $[cover]$ *of arity* `Arrow`*, but we could also create compound labels like* $([card], (m_1, m_2))$ *and* $([card], (n_1, n_2))$ *of arity* `Arrow` *to describe arbitrary intervals for the cardinalities of* $f$ *and* $g$*, respectively.*

*The difference between an association and an association class is that an association class represents explicitly also the relation* $R$*. Therefore, we should include into* $\Theta_{\mathbf{UML}}$ *the label* $([1{-}1], 2)$ *with arity* `Span`*. We should also add to the signature a label* $[graph]$ *with arity* `Triangle` *to express that* $f$ *or* $g$ *represent the "graph" of* $R$*.* □

[8] The current version of the standard, UML2, defines a rather general and complex notion of associations, details and a formal semantics for it can be found in [DD06].

In the process of a stepwise design we need to extend specifications, rename items, identify items. In this way we produce a sequence of related specifications. Those relations are described by

**Definition 6** (Sketch Morphisms)**.** *A* $\Theta$***-sketch morphism*** $f : \mathcal{S} \to \mathcal{S}'$ *between two* $\Theta$*-sketches* $\mathcal{S} = (G^{\mathcal{S}}, \mathcal{S}(\Pi))$ *and* $\mathcal{S}' = (G^{\mathcal{S}'}, \mathcal{S}'(\Pi))$ *is a graph homomorphism* $f : G^{\mathcal{S}} \to G^{\mathcal{S}'}$ *compatible with marked diagrams, i.e.,* $(P, \delta : \alpha(P) \to G^{\mathcal{S}}) \in \mathcal{S}(P)$ *implies* $(P, \delta; f : \alpha(P) \to G^{\mathcal{S}'}) \in \mathcal{S}'(P)$ *for all* $P \in \Pi$*.*




**Remark 6** (Subsketches)**.** *Note, that* $\mathcal{S} \sqsubseteq \mathcal{T}$ *iff the inclusion graph homomorphism* $in : G^{\mathcal{S}} \hookrightarrow G^{\mathcal{T}}$ *defines a* $\Theta$*-sketch homomorphism* $in : \mathcal{S} \hookrightarrow \mathcal{T}$*.*

**Remark 7** (Category of Sketches)**.** *The associativity of the composition of graph homomorphisms ensures that the composition of two* $\Theta$*-sketch morphisms becomes a* $\Theta$*-sketch morphism as well, and that the composition of* $\Theta$*-sketch morphisms is associative too. Further the identical graph homomorphism* $id_{G^{\mathcal{S}}} = (id_{G_0^{\mathcal{S}}}, id_{G_1^{\mathcal{S}}}) : G^{\mathcal{S}} \to G^{\mathcal{S}}$ *defines an identical* $\Theta$*-sketch morphism* $id_{\mathcal{S}} : \mathcal{S} \to \mathcal{S}$ *that is neutral w.r.t. the composition of* $\Theta$*-sketch morphisms. In such a way, we obtain for any diagram signature* $\Theta$ *a category* $\mathbf{Ske}(\Theta)$*.*

Another situation, we often meet in "industrial" projects, is that we have to extend the specification formalism because we realize that some relevant system aspects can not be specified by our formalism of choice, Or, even worse, the "industrial partner" may force us to switch to another formalism for some "objective reasons". If the involved formalisms are described according the Generalized Sketch pattern (some of) those changes can be reflected by signature morphisms.

**Definition 7** (Signature Morphism)**.** *A* ***signature morphism*** $\varphi : \Theta_1 \to \Theta_2$ *between two diagram signatures* $\Theta_1 = (\Pi_1, \alpha_1)$ *and* $\Theta_2 = (\Pi_2, \alpha_2)$ *is a map* $\varphi : \Pi_1 \to \Pi_2$ *such that* $\alpha_2(\varphi(P_1)) = \alpha_1(P_1)$ *for each* $P_1 \in \Pi_1$*.*

Based on the concept of signature morphism we can attack, describe, and investigate questions like relations between different formalisms, translation of specifications, heterogeneous (multi-paradigm) specifications, integration of formalisms and so on. The re-labeling of diagrams provides a translation of specifications.

**Proposition 1.** *Any signature morphism* $\varphi : \Theta_1 \to \Theta_2$ *induces a functor* $\varphi^* : \mathbf{Ske}(\Theta_2) \to \mathbf{Ske}(\Theta_1)$*.*

*Proof.* Any $\Theta_2$-sketch $\mathcal{S}_2 = (G^{\mathcal{S}_2}, \mathcal{S}_2(\Pi_2))$ can be translated into a $\Theta_1$-sketch $\varphi^*(\mathcal{S}_2) = (G^{\mathcal{S}_2}, \varphi^*(\mathcal{S}_2)(\Pi_1))$ where $\varphi^*(\mathcal{S}_2)(P_1) \stackrel{def}{=} \mathcal{S}_2(\varphi(P_1))$ for all $P_1 \in \Pi_1$, i.e., we have

$$(P_1, \delta) \in \varphi^*(\mathcal{S}_2)(P_1) \quad \textbf{iff} \quad (\varphi(P_1), \delta) \in \mathcal{S}_2(\varphi(P_1)).$$

This is well-defined since $\alpha_2(\varphi(P_1)) = \alpha_1(P_1)$.

A $\Theta_2$-sketch morphism $f : \mathcal{S}_2 \to \mathcal{S}_2'$ is given by a graph homomorphism $f : G^{\mathcal{S}_2} \to G^{\mathcal{S}_2'}$ such that $(P_2, \delta) \in \mathcal{S}_2(P_2)$ implies $(P_2, \delta; f) \in \mathcal{S}_2'(P_2)$ for all $P_2 \in \Pi_2$. This entails for any $P_1 \in \Pi_1$:

$$\begin{array}{lll} & (P_1, \delta) \in \varphi^*(\mathcal{S}_2)(P_1) & \\ \Leftrightarrow & (\varphi(P_1), \delta) \in \mathcal{S}_2(\varphi(P_1)) & \text{(def. of } \varphi^*(\mathcal{S}_2)) \\ \Rightarrow & (\varphi(P_1), \delta; f) \in \mathcal{S}_2'(\varphi(P_1)) & (f \text{ is } \Theta_2\text{-sketch morphism }) \\ \Leftrightarrow & (P_1, \delta; f) \in \varphi^*(\mathcal{S}_2')(P_1) & \text{(def. of } \varphi^*(\mathcal{S}_2')) \end{array}$$

This means, however, that the graph homomorphism $f : G^{\mathcal{S}_2} \to G^{\mathcal{S}_2'}$ defines also a $\Theta_1$-sketch morphism $\varphi^*(f) \stackrel{def}{=} f : \varphi^*(\mathcal{S}_2) \to \varphi^*(\mathcal{S}_2')$. Compatibility of $\varphi^*$ w.r.t. identities and composition follows immediately from definition. □

For non-injective signature morphisms $\varphi$ the functor $\varphi^*$ produces copies of diagrams with different labels. On the other side, non-injective signature morphisms can be considered as a request to identify different labels. Such a identification of labels induces a corresponding identification of diagrams.

**Proposition 2.** *Any signature morphism* $\varphi : \Theta_1 \to \Theta_2$ *induces a functor* $\varphi_* : \mathbf{Ske}(\Theta_1) \to \mathbf{Ske}(\Theta_2)$*.*

*Proof.* Any $\Theta_1$-sketch $\mathcal{S}_1 = (G^{\mathcal{S}_1}, \mathcal{S}_1(\Pi_1))$ can be translated into a $\Theta_2$-sketch $\varphi_*(\mathcal{S}_1) = (G^{\mathcal{S}_1}, \varphi_*(\mathcal{S}_1)(\Pi_2))$ where $\varphi_*(\mathcal{S}_1)(P_2) \stackrel{def}{=} \bigcup\{\mathcal{S}_1(P_1) \mid P_1 \in \Pi_1, \varphi(P_1) = P_2\}$ for all $P_2 \in \Pi_2$, i.e., we have

$$(P_2, \delta) \in \varphi_*(\mathcal{S}_1)(P_2) \quad \textbf{iff} \quad \text{there exists a } P_1 \in \Pi_1 \text{ such that } \varphi(P_1) = P_2 \text{ and } (P_1, \delta) \in \mathcal{S}_1(P_1).$$

This is well-defined since $\alpha_2(\varphi(P_1)) = \alpha_1(P_1)$.

A $\Theta_1$-sketch morphism $f : \mathcal{S}_1 \to \mathcal{S}_1'$ is given by a graph homomorphism $f : G^{\mathcal{S}_1} \to G^{\mathcal{S}_1'}$ such that $(P_1, \delta) \in \mathcal{S}_1(P_1)$ implies $(P_1, \delta; f) \in \mathcal{S}_1'(P_1)$ for all $P_1 \in \Pi_1$. This entails for any $P_2 \in \varphi(\Pi_1)$ :

$$\begin{array}{lll} & (P_2, \delta) \in \varphi_*(\mathcal{S}_1)(P_2) & \\ \Leftrightarrow & (P_1, \delta) \in \mathcal{S}_1(P_1) \text{ for a } P_1 \in \Pi_1 \text{ with } \varphi(P_1) = P_2 & \text{(def. of } \varphi_*(\mathcal{S}_1)) \\ \Rightarrow & (P_1, \delta; f) \in \mathcal{S}_1'(P_1) \text{ for a } P_1 \in \Pi_1 \text{ with } \varphi(P_1) = P_2 & (f \text{ is } \Theta_1\text{-sketch morphism }) \\ \Leftrightarrow & (P_2, \delta; f) \in \varphi_*(\mathcal{S}_1)(P_2) & \text{(def. of } \varphi_*(\mathcal{S}_1')) \end{array}$$

For all $P_2 \in \Pi_2 \setminus \varphi(\Pi_1)$ we have $\varphi_*(\mathcal{S}_1)(P_2) = \emptyset$, thus the morphism condition is trivially satisfied for all these "new" predicate symbols. This shows, finally, that the graph homomorphism $f : G^{\mathcal{S}_1} \to G^{\mathcal{S}_1'}$ defines also a $\Theta_2$-sketch morphism $\varphi_*(f) \stackrel{def}{=} f : \varphi_*(\mathcal{S}_1) \to \varphi_*(\mathcal{S}_1')$. Compatibility of $\varphi_*$ w.r.t. identities and composition follows immediately from definition. □

The intuition of $\varphi^*$ and $\varphi_*$ being a kind of inverse constructions can be expressed formally.

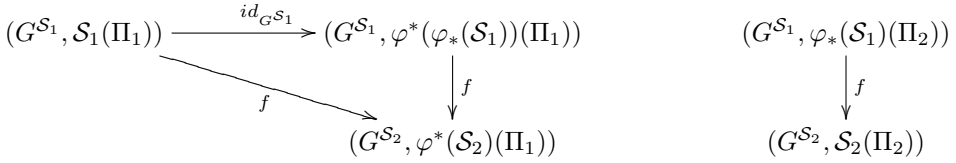
**Proposition 3.** *For any signature morphism $\varphi : \Theta_1 \to \Theta_2$ the functor $\varphi_* : \mathbf{Ske}(\Theta_1) \to \mathbf{Ske}(\Theta_2)$ is left-adjoint to the functor $\varphi^* : \mathbf{Ske}(\Theta_2) \to \mathbf{Ske}(\Theta_1)$.*

*Proof.* **Unit:** For any $\Theta_1$-sketch $\mathcal{S}_1 = (G^{\mathcal{S}_1}, \mathcal{S}_1(\Pi_1))$ and any $P_1 \in \Pi_1$ we have according our definitions

$$\begin{array}{lll} \mathcal{S}_1(P_1) & \subseteq & \bigcup\{\mathcal{S}_1(P_1') \mid P_1' \in \Pi_1, \varphi(P_1') = \varphi(P_1)\} \\ & = & \varphi_*(\mathcal{S}_1)(\varphi(P_1)) \\ & = & \varphi^*(\varphi_*(\mathcal{S}_1))(P_1) \end{array}$$

This ensures that the identical graph homomorphism $id_{G^{\mathcal{S}_1}}$ defines a $\Theta_1$-sketch morphism from $\mathcal{S}_1$ to $\varphi^*(\varphi_*(\mathcal{S}_1))$.

$$\mathbf{Ske}(\Theta_1) \underset{\varphi^*}{\overset{\varphi_*}{\rightleftarrows}} \mathbf{Ske}(\Theta_2)$$




For any $\Theta_2$-sketch $\mathcal{S}_2 = (G^{\mathcal{S}_2}, \mathcal{S}_2(\Pi_2))$ and any $\Theta_1$-sketch morphism $f : \mathcal{S}_1 \to \varphi^*(\mathcal{S}_2)$ we can show that the underlying graph homomorphism $f : G^{\mathcal{S}_1} \to G^{\mathcal{S}_2}$ provides also a $\Theta_2$-sketch morphism $f : \varphi_*(\mathcal{S}_1) \to \mathcal{S}_2$: For any $P_2 \in \Pi_2 \setminus \varphi(\Pi_1)$ we have $\varphi_*(\mathcal{S}_1)(P_2) = \emptyset$, thus the morphism condition is trivially satisfied. For any $P_2 \in \varphi(\Pi_1)$ and any $(P_2, \delta) \in \varphi_*(\mathcal{S}_1)(P_2)$ there exists $P_1 \in \Pi_1$ such that $\varphi(P_1) = P_2$ and $(P_1, \delta) \in \mathcal{S}_1(P_1)$. $f$ a $\Theta_1$-sketch morphism implies $(P_1, \delta; f) \in \varphi^*(\mathcal{S}_2)(P_1)$. Due to the definition of $\varphi^*(\mathcal{S}_2)$ this means, however, $(P_2, \delta; f) = (\varphi(P_1), \delta; f) \in \mathcal{S}_2(\varphi(P_1)) = \mathcal{S}_2(P_2)$ as required.

Finally, $f : \varphi_*(\mathcal{S}_1) \to \mathcal{S}_2$ is uniquely determined since $f$ is the only graph homomorphism such that $id_{G^{\mathcal{S}_1}}; f = f$. □

# 4 Models of Generalized Sketches

To define models of $\Theta$-sketches we have first to decide for a **"semantic universe"**, i.e., we have to chose an appropriate "big" graph $U$. In case of algebraic and first-order specifications we could chose, for example, the graph Set. For ER-diagrams the graph Par and for UML class diagrams the graph Pow may be appropriated at the beginning, i.e., as long as we are not concernd about the variation of "states" in time. (In [DK03], the semantic

universe of "variable sets" is presented that can be used to describe the dynamics of systems.) In a second step we have to convert the graph $U$ into a "semantic $\Theta$-sketch" $\mathcal{U} = (U, \mathcal{U}(\Pi))$, i.e., for all labels $P \in \Pi$ we have to give a mathematical exact definition of the "property $P$", i.e., of the set $\mathcal{U}(P)$ of all "semantic diagrams" of arity $\alpha(P)$. Then we can define models within this chosen "semantic universe".

But, if we want to have also morphisms between models we need some more structure in the "semantic universe". One possibility to define morpisms between models is to adapt the concept of "natural transformation" [BW95, Fia05] and, in this case, it will be necessary that the arrows in the "semantic universe" can be composed. That is, we have to assume that the graph $U$ is actually a category. Moreover, it appears to be often necessary, in practice, to allow only for a certain kind of arrows to relate the components of models, That is, we have to indicate an appropriate subcategory of $U$.

**Definition 8** (Semantic Universe)**.** *A **semantic $\Theta$-universe** $\mathcal{U} = (\mathsf{U}, \mathcal{U}(\Pi), \mathsf{U}^m)$ is given by a category $\mathsf{U}$ which is the underlying graph of a $\Theta$-sketch $(\mathsf{U}, \mathcal{U}(\Pi))$ and by a subgategory $\mathsf{U}^m$ of $\mathsf{U}$ such that $\mathsf{U}^m_0 = \mathsf{U}_0$. (In abuse of notation we will denote the $\Theta$-sketch $(\mathsf{U}, \mathcal{U}(\Pi))$ also by $\mathcal{U}$.)*

**Example 9** (Semantic Universes)**.** *In case of **algebraic specifications** we can take as the semantic $\Theta_{\mathbf{AS}}$-universe the triple $\mathcal{AS} = (\mathsf{Set}, \mathcal{AS}(\Pi_{\mathbf{AS}}), \mathsf{Set})$. All singleton sets will be marked with the "empty product" label $([prod], 0)$, and the labels $([prod], n)$, $n \geq 2$ are marking not only the cartesian products of $n$ sets, but also all respective isomorphic diagrams.*

*For **first-order signatures** we can extend this universe to a semantic $\Theta_{\mathbf{FS}}$-universe $\mathcal{FS} = (\mathsf{Set}, \mathcal{FS}(\Pi_{\mathbf{FS}}), \mathsf{Set})$ by marking all injective maps and all jointly injective families of maps, respectively. Note, that "marking" means here to give a precise mathematical description of the corresponding property.*

*For **equational specifications** we can extend $\mathcal{AS}$ to a semantic $\Theta_{\mathbf{EQ}}$-universe $\mathcal{EQ} = (\mathsf{Set}, \mathcal{EQ}(\Pi_{\mathbf{EQ}}), \mathsf{Set})$. $\mathcal{EQ}([=])$ is the set of all diagrams $\delta : \mathtt{Cell} \to \mathsf{Set}$ with $\delta_1(1) = \delta_1(2)$. $\mathcal{EQ}([comp])$ is the set of all diagrams $\delta : \mathtt{Triangle} \to \mathsf{Set}$ with $\delta_1(3) = \delta_1(1); \delta_1(2)$. $\mathcal{EQ}([tupl], n)$ is the set of all diagrams $\delta : \mathtt{Tupl}_n \to \mathsf{Set}$ with $\delta_1(\pi_i)(\delta_1(k)(e)) = \delta_1(r_i)(e)$ for all $e \in \delta_0(y)$, $i = 1, \ldots, n$. And $\mathcal{EQ}([id])$ is the set of all diagrams $\delta : \mathtt{Arrow} \to \mathsf{Set}$ with $\delta_0(x) = \delta_0(y)$ and $\delta_1(1) = id_{\delta_0(x)}$.*

*In case of partial algebras [Rei87, Wol90, WKWC94] we have to use the triple $\mathcal{EQ}_p = (\mathsf{Par}, \mathcal{EQ}_p(\Pi_{\mathbf{EQ}}), \mathsf{Set})$ since homomorphisms between partial algebras are given by total maps.*

*For **ER diagrams** a semantic $\Theta_{\mathbf{ER}}$-universe can be based on the category $\mathsf{Par}$ or on the category $\mathsf{Pow}$ for those variants of ER diagrams where we allow for multiple attributes. And for **UML class diagrams** a semantic $\Theta_{\mathbf{UML}}$-universe should be based on the category $\mathsf{Pow}$.* □

The requirement, that a predicate in a specification has to become true under a semantical interpretation can be formulated now by means of the concept of sketch morphism.

**Definition 9** (Models)**.** *An **$\mathcal{S}$-model** of a $\Theta$-sketch $\mathcal{S}$ in a semantic $\Theta$-universe $\mathcal{U}$ is a $\Theta$-sketch morphism $m : \mathcal{S} \to \mathcal{U}$, i.e., $(P, \delta : \alpha(P) \to G^{\mathcal{S}}) \in \mathcal{S}(P)$ implies $(P, \delta; m : \alpha(P) \to \mathsf{U}) \in \mathcal{U}(P)$ for all $P \in \Pi$.*

$$\begin{array}{ccc} \alpha(P) & \xrightarrow{\delta} & G^{\mathcal{S}} \\ & \searrow^{\delta;m} & \downarrow{\scriptstyle m} \\ & & \mathsf{U} \end{array}$$

*An **$\mathcal{S}$-model morphism** $\eta : m_1 \Rightarrow m_2$ between two $\mathcal{S}$-models $m_1 : \mathcal{S} \to \mathcal{U}$ and $m_2 : \mathcal{S} \to \mathcal{U}$ is given by a map $\eta : G^{\mathcal{S}}_0 \to \mathsf{U}^m_1$ such that $\eta(x); m_2(f) = m_1(f); \eta(y)$ for each $f : x \to y$ in $G^{\mathcal{S}}_1$.*

$$\begin{array}{ccccc} x & & m_1(x) & \xrightarrow{\eta(x)} & m_2(x) \\ \downarrow{\scriptstyle f} & & \downarrow{\scriptstyle m_1(f)} & & \downarrow{\scriptstyle m_2(f)} \\ y & & m_1(y) & \xrightarrow{\eta(y)} & m_2(y) \end{array}$$

**Example 10** (Models)**.** *Models of $\Theta_{\mathbf{AS}}$-sketches and $\Theta_{\mathbf{EQ}}$-sketches correspond to $\Sigma$-algebras and the traditional homomorphisms between $\Sigma$-algebras are reflected by model morphisms. We can define, for example, the "natural numbers" as a $\mathcal{S}_{\Sigma_{Nat}}$-model $nat : \mathcal{S}_{\Sigma_{Nat}} \to \mathcal{AS}$ given by $nat_0(N) = \mathbb{N}$, $nat_0(M) = \mathbb{N} \times \mathbb{N}$, $nat_1(z) = 0$, $nat_1(s) = (\_ + 1)$, $nat_1(p) = (\_ + \_)$, and two projections $nat_1(\pi_i) : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, $i = 1, 2$. Since the*

*equation $(p(x,s(y)) = s(p(x,y))$ is satisfied for natural numbers this $\mathcal{S}_{\Sigma_{Nat}}$-model can be extended uniquely to a $\mathcal{S}_{SP_{Nat}}$-model.*

*"Binary digits" give rise to another $\mathcal{S}_{\Sigma_{Nat}}$-model $bin : \mathcal{S}_{\Sigma_{Nat}} \to \mathcal{AS}$ given by $bin_0(N) = \mathbf{2} = \{0,1\}$, $bin_0(M) = \mathbf{2} \times \mathbf{2}$, $bin_1(z) = 0$, $bin_1(s) = (\_ + 1 \bmod 2)$, $bin_1(p) = (\_ + \_ \bmod 2)$, and two projections $bin_1(\pi_i) : \mathbf{2} \times \mathbf{2} \to \mathbf{2}$, $i = 1,2$. Then the maps $(\mathsf{mod}\ 2) : \mathbb{N} \to \mathbf{2}$ and $(\mathsf{mod}\ 2) \times (\mathsf{mod}\ 2) : \mathbb{N} \times \mathbb{N} \to \mathbf{2} \times \mathbf{2}$ define a $\mathcal{S}_{\Sigma_{Nat}}$-model morphism from $nat$ to $bin$.*

*Note, that models of $\Theta_{\mathbf{ER}}$-sketches and $\Theta_{\mathbf{UML}}$-sketches, respectively, correspond to "system states" as long as the corresponding semantic universes are based on* Par *or* Pow*, respectively. A well-known observation is that "state transformations" can* ***not*** *be described by model morphisms. One possible way is to describe those transformations by spans of model morphisms [JR01]. A detailed investigation of the possibilities to model "state transformations" will be another point of future research.* □

In the same way as one can prove that functors and natural transformations constitute a "functor category" (see [BW95]) we could prove the existence of a

**Proposition 4** (Category of Models)**.** *The* ***$\mathcal{S}$-models*** *$m : \mathcal{S} \to \mathcal{U}$ of a $\Theta$-sketch $\mathcal{S}$ in a semantic $\Theta$-universe $\mathcal{U}$ and the* ***$\mathcal{S}$-model morphisms*** *$\eta : m_1 \Rightarrow m_2 : \mathcal{S} \to \mathcal{U}$ define a category $Mod(\mathcal{S},\mathcal{U})$.*

*The* ***identical $\mathcal{S}$-model morphisms*** *$id_m : m \Rightarrow m$ are defined by $id_m(x) \stackrel{def}{=} id_{m(x)}$ for all $x \in G_0^{\mathcal{S}}$. And the composition $\eta;\mu : m_1 \Rightarrow m_3$ of two $\mathcal{S}$-model morphisms $\eta : m_1 \Rightarrow m_2$ and $\mu : m_2 \Rightarrow m_3$ is defined by $(\eta;\mu)(x) \stackrel{def}{=} \eta(x);\mu(x)$ for all $x \in G_0^{\mathcal{S}}$.*

$$\begin{array}{ccccc} x & & m_1(x) & \xrightarrow{\eta(x)} & m_2(x) \xrightarrow{\mu(x)} m_3(x) \\ {\scriptstyle f}\downarrow & & {\scriptstyle m_1(f)}\downarrow & & \downarrow{\scriptstyle m_2(f)} \qquad \downarrow{\scriptstyle m_3(f)} \\ y & & m_1(y) & \xrightarrow{\eta(y)} & m_2(y) \xrightarrow{\mu(y)} m_3(y) \end{array}$$

**Remark 8** (Other Model Morphisms)**.** *The commutativity requirement in Proposition 4 may appear too strong in some applications. To sketch, for example, "weak homomorphisms" between partial algebras [Rei87, Wol90] we have to relax the equality into an inequality $(\eta;\mu)(x) \leq \eta(x);\mu(x)$ expressing that the definedness of operations needs only to be preserved.*

*In other applications, it may be necessary to relate models by more general arrows. This variation, together with all forthcoming results, can be simply obtained by requiring that* $\mathsf{U}$ *is a subcategory of* $\mathsf{U}^m$*.*

Since all our concepts are defined in a clean categorical way we get, for example, different kinds of model transformations for free. First, any "specification morphism" gives rise, just by pre-composition, to a transformation of models into the opposite direction.

**Proposition 5** (Forgetful Functor)**.** *For any $\Theta$-sketch morphism $f : \mathcal{S} \to \mathcal{S}'$ we obtain a (forgetful) functor $Mod(f) : Mod(\mathcal{S}',\mathcal{U}) \to Mod(\mathcal{S},\mathcal{U})$ where $Mod(f)(m) \stackrel{def}{=} f;m$ for any $\mathcal{S}'$-model $m : \mathcal{S}' \to \mathcal{U}$ and $Mod(f)(\eta) \stackrel{def}{=} f_0;\eta$ for any $\mathcal{S}'$-model morphism $\eta : m \Rightarrow m'$.*

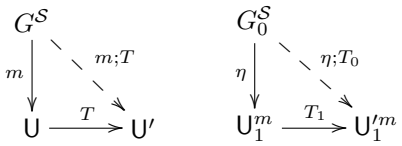
$$\begin{array}{ccc} G^{\mathcal{S}} & \xrightarrow{f} & G^{\mathcal{S}'} \\ & {\scriptstyle f;m}\searrow & \downarrow{\scriptstyle m} \\ & & \mathsf{U} \end{array} \qquad \begin{array}{ccc} G_0^{\mathcal{S}} & \xrightarrow{f_0} & G_0^{\mathcal{S}'} \\ & {\scriptstyle f_0;\eta}\searrow & \downarrow{\scriptstyle \eta} \\ & & \mathsf{U}_1^m \end{array}$$

At a certain point of a development it may be necessary that we have to extend our semantic universe to be able to describe phenomena that can not be reflected within the given universe. We may want, for example, describe the dynamic aspects of systems after we have clarified the static aspects. Those extensions or transformations of semantic universes can be described by

**Definition 10** (Transformation of Universes)**.** *A* ***$\Theta$-transformation*** *$T : \mathcal{U} \to \mathcal{U}'$ between two semantic $\Theta$-universes $\mathcal{U} = (\mathsf{U},\mathcal{U}(\Pi),\mathsf{U}^m)$ and $\mathcal{U}' = (\mathsf{U}',\mathcal{U}'(\Pi),\mathsf{U}'^m)$ is given by a functor $T : \mathsf{U} \to \mathsf{U}'$ with $T(\mathsf{U}^m) \subseteq \mathsf{U}'^m$ that defines a $\Theta$-sketch morphism $T : (\mathsf{U},\mathcal{U}(\Pi)) \to (\mathsf{U}',\mathcal{U}'(\Pi))$.*

Any transformation of universes induces a corresponding transformation of models by a simple post-composition:

**Proposition 6** (Transformation of Models)**.** *A $\Theta$-transformation $T : \mathcal{U} \to \mathcal{U}'$ between semantic $\Theta$-universes induces for any $\Theta$-sketch $\mathcal{S}$ a functor $Mod(\mathcal{S},T) : Mod(\mathcal{S},\mathcal{U}) \to Mod(\mathcal{S},\mathcal{U}')$ where $Mod(\mathcal{S},T)(m) \stackrel{def}{=} m;T$ for any $\mathcal{S}$-model $m : \mathcal{S} \to \mathcal{U}$ and $Mod(\mathcal{S},T)(\eta) \stackrel{def}{=} \eta;T_1$ for any $\mathcal{S}$-model morphism $\eta : m_1 \Rightarrow m_2 : \mathcal{S} \to \mathcal{U}$.*




How can we transform now a stepwise and structured design from one formalism $(\Theta_1,\mathcal{U})$ to another formalism $(\Theta_2,\mathcal{U}')$? First, we need a signature morphims $\varphi : \Theta_1 \to \Theta_2$. This allows us, on one side, to interpret the $\Theta_1$-universe $\mathcal{U}$ as a $\Theta_2$-universe according to Proposition 2. That is, we obtain a $\Theta_2$-universe $\varphi_*(\mathcal{U}) = (\mathsf{U},\varphi_*(\mathcal{U})(\Pi_2),\mathsf{U}^m)$ and, by restricting $\varphi_* : \mathbf{Ske}(\Theta_1) \to \mathbf{Ske}(\Theta_2)$ to $Mod(\mathcal{S},\mathcal{U})$ we obtain for any $\Theta_1$-sketch $\mathcal{S}$ a functor $\varphi_{*,\mathcal{S}} : Mod(\mathcal{S},\mathcal{U}) \to Mod(\varphi_*(\mathcal{S}),\varphi_*(\mathcal{U}))$ that transforms $\mathcal{S}$-models over $\mathcal{U}$ into $\varphi_*(\mathcal{S})$-models over $\varphi_*(\mathcal{U})$. In a second step, we have to adapt the models to the richer new universe. That is, we need a $\Theta_2$-transformation $T : \varphi_*(\mathcal{U}) \to \mathcal{U}'$. The whole transformation from $(\Theta_1,\mathcal{U})$ to $(\Theta_2,\mathcal{U}')$ can be described then for a given $\Theta_1$-sketch $\mathcal{S}$ by a composition of functors

$$\varphi_{*,\mathcal{S}};Mod(\varphi_*(\mathcal{S}),T) : Mod(\mathcal{S},\mathcal{U}) \to Mod(\varphi_*(\mathcal{S}),\mathcal{U}').$$

The interested reader may check that we obtain, in such a way, for any $\Theta_1$-sketch morphism $f : \mathcal{S}_1 \to \mathcal{S}_1'$ a commutative diagram

$$\begin{array}{ccc} Mod(\mathcal{S}_1',\mathcal{U}) & \xrightarrow{Mod(f)} & Mod(\mathcal{S}_1,\mathcal{U}) \\ \downarrow{\scriptstyle \varphi_{*,\mathcal{S}_1'};Mod(\varphi_*(\mathcal{S}_1'),T)} & & \downarrow{\scriptstyle \varphi_{*,\mathcal{S}_1};Mod(\varphi_*(\mathcal{S}_1),T)} \\ Mod(\varphi_*(\mathcal{S}_1'),\mathcal{U}') & \xrightarrow{Mod(\varphi_*(f))} & Mod(\varphi_*(\mathcal{S}_1),\mathcal{U}') \end{array}$$

That is, we can not only transform the single specifications but also the structure of our design.

# 5 Sketch Operations

Sketch Operations provide a mechanism to extend sketches, in a well-defined constructive way, by deriving new nodes, new arrows, and new marked diagrams. Such a mechanism allows us to reflect the construction of "algebraic terms" or deduction rules within the Generalized Sketch framework [Mak97][Dis97b] and, for example, to define the semantics of database query languages and view mechanisms [DK97]. Derived structures are also necessary to relate specifications and to describe data and schema integration and other model management tasks [Dis05].

**Definition 11** (Sketch Operation)**.** *A $\Theta$-**sketch operation** $\omega : \mathcal{L} \hookrightarrow \mathcal{R}$ is given by two $\Theta$-sketches $\mathcal{L} = (G^{\mathcal{L}},\mathcal{L}(\Pi))$, $\mathcal{R} = (G^{\mathcal{R}},\mathcal{R}(\Pi))$ such that $\mathcal{L} \sqsubseteq \mathcal{R}$.*

The left hand side $\mathcal{L}$ specifies under what conditions the sketch operation can be applied to a sketch $\mathcal{S}$. The idea is to add to $\mathcal{S}$ (a copy of) the items in $\mathcal{R}$ that are not contained in $\mathcal{L}$. Analogously to programming, $\mathcal{L}$ can be seen as a "formal parameter" where an "actual parameter" is obtained by assigning to the "variable items" in $\mathcal{L}$ "actual items" from a given $\mathcal{S}$. First, we describe how sketch operations can be used, on the "syntactic level", to extend specifications.

**Definition 12** (Syntactic Sketch Operations)**.** *A **match** of a $\Theta$-sketch operation $\omega : \mathcal{L} \hookrightarrow \mathcal{R}$ in a $\Theta$-sketch $\mathcal{S} = (G^{\mathcal{S}},\mathcal{S}(\Pi))$ is given by a $\Theta$-sketch morphism $a : \mathcal{L} \to \mathcal{S}$.*

*The **application** of the $\Theta$-sketch operation $\omega : \mathcal{L} \hookrightarrow \mathcal{R}$ to a $\Theta$-sketch $\mathcal{S}$ **via a match** $a : \mathcal{L} \to \mathcal{S}$ results in a $\Theta$-sketch $\omega(a) = (G^{\omega(a)},\omega(a)(\Pi))$ and in a $\Theta$-sketch morphism $a^* : \mathcal{R} \to \omega(a)$ such that $\mathcal{S} \sqsubseteq \omega(a)$ and $a;\sqsubseteq = \sqsubseteq;a^*$.*

$$\begin{array}{ccc} \mathcal{L} & \xrightarrow{\sqsubseteq} & \mathcal{R} \\ \downarrow{\scriptstyle a} & & \downarrow{\scriptstyle a^*} \\ \mathcal{S} & \xrightarrow{\sqsubseteq} & \omega(a) \end{array}$$

*Thereby the underlying graph $G^{\omega(a)}$ is defined as follows*

$$G_i^{\omega(a)} \stackrel{def}{=} G_i^{\mathcal{S}} \cup \{(a,x) \mid x \in G_i^{\mathcal{R}} \setminus G_i^{\mathcal{L}}\} \quad i = 0,1.$$

$$sc^{G^{\omega(a)}}(f) \stackrel{def}{=} \begin{cases} sc^{G^{\mathcal{S}}}(f) & ,\text{if } f \in G_1^{\mathcal{S}} \\ a_0(sc^{G^{\mathcal{R}}}(g)), \text{if } f = (a,g), sc^{G^{\mathcal{R}}}(g) \in G_0^{\mathcal{L}} \\ (a, sc^{G^{\mathcal{R}}}(g)), \text{if } f = (a,g), sc^{G^{\mathcal{R}}}(g) \notin G_0^{\mathcal{L}} \end{cases}$$

$$tg^{G^{\omega(a)}}(f) \stackrel{def}{=} \begin{cases} tg^{G^{\mathcal{S}}}(f) & ,\text{if } f \in G_1^{\mathcal{S}} \\ a_0(tg^{G^{\mathcal{R}}}(g)), \text{if } f = (a,g), tg^{G^{\mathcal{R}}}(g) \in G_0^{\mathcal{L}} \\ (a, tg^{G^{\mathcal{R}}}(g)), \text{if } f = (a,g), tg^{G^{\mathcal{R}}}(g) \notin G_0^{\mathcal{L}} \end{cases}$$

*The graph homomorphism $a^* : G^{\mathcal{R}} \to G^{\omega(a)}$ is given for $i = 0,1$ by*

$$a_i^*(x) \stackrel{def}{=} \begin{cases} a_i(x), \text{if } x \in G_i^{\mathcal{L}} \\ (a,x), \text{if } x \in G_i^{\mathcal{R}} \setminus G_i^{\mathcal{L}} \end{cases}$$

*And for any $P \in \Pi$ we have* $\omega(a)(P) \stackrel{def}{=} \mathcal{S}(P) \cup \{(P,\delta;a^*) \mid (P,\delta) \in \mathcal{R}(P) \setminus \mathcal{L}(P)\}$.

Note, that $a^* : G^{\mathcal{R}} \to G^{\omega(a)}$ defines indeed a $\Theta$-sketch morphism $a^* : \mathcal{R} \to \omega(a)$: For all $(P,\delta) \in \mathcal{L}(P)$ we have $(P,\delta;a^*) = (P,\delta;a)$ due to the definition of $a^*$ and thus $(P,\delta;a^*) = (P,\delta;a) \in \mathcal{S}(P) \subseteq \omega(a)(P)$ since $a : \mathcal{L} \to \mathcal{S}$ is a $\Theta$-sketch morphism. For all $(P,\delta) \in \mathcal{R}(P) \setminus \mathcal{L}(P)$ we obtain directly $(P,\delta;a^*) \in \omega(a)(P)$ due to the definition of $\omega(a)(P)$.

**Remark 9** (Disjoint Union)**.** *To describe the disjoint union of $\mathcal{S}$ and $\mathcal{R} \setminus \mathcal{L}$ we have used the mechanism of "tagging" the added items with the corresponding match. This ensures, for example, that different applications of the same sketch operation will produce different copies of $\mathcal{R} \setminus \mathcal{L}$. Of course, we can also adapt, in practical applications, more advanced mechanisms to create names/labels for added items. See also the discussion in Remark 11.*

**Remark 10** (Pushout)**.** *The attentive reader, familiar with Category Theory and Graph Transformations [EEPT06, Hec06], will have noticed that the construction in Definition 12 is actually a pushout construction, i.e., can be seen as a special simple version of a graph transformation. The specialization of the general theory of graph transformations to this special case will be the subject of further research. Especially the sequential and parallel composition of sketch operations will be of interest.*

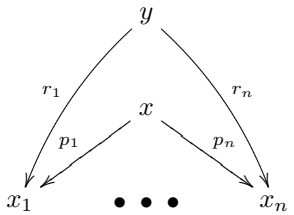
**Example 11** (Term Construction)**.** *The inductive construction of $\Sigma$-terms over finite sets of variables can be modeled by $\Theta_{\mathbf{EQ}}$-sketch operations. The necessary operations are the operation $[id\rangle : \mathcal{L}_{[id\rangle} \hookrightarrow \mathcal{R}_{[id\rangle}$ introducing identity requirements, the operation $[comp\rangle : \mathcal{L}_{[comp\rangle} \hookrightarrow \mathcal{R}_{[comp\rangle}$ generating the composition of two arrows, the operations $([prod\rangle, n) : \mathcal{L}_{([prod\rangle,n)} \hookrightarrow \mathcal{R}_{([prod\rangle,n)}$ introducing products of $n$ nodes, and the operations $([tupl\rangle, n) : \mathcal{L}_{([tupl\rangle,n)} \hookrightarrow \mathcal{R}_{([tupl\rangle,n)}$ describing the **tupling** of $n$ arrows.*

*$\mathcal{L}_{[id\rangle}$ is given by the discrete graph $(x\ y)$ and $\mathcal{R}_{[id\rangle}$ is given by the graph* `Arrow` *and the $[id]$-marked diagram $id_{\texttt{Arrow}}$.*

*$\mathcal{L}_{[comp\rangle}$ is given by the graph $(x \xrightarrow{1} y \xrightarrow{2} z)$ and $\mathcal{R}_{[comp\rangle}$ is given by the graph* `Triangle` *and the $[comp]$-marked diagram $id_{\texttt{Triangle}}$.*

*$\mathcal{L}_{([prod\rangle,n)}$ is given by the discrete graph $(x_1 \dots x_n)$ and $\mathcal{R}_{([prod\rangle,n)}$ is given by the graph* $\texttt{Span}_n$ *and the $([prod], n)$-marked diagram $id_{\texttt{Span}_n}$.*

*$\mathcal{L}_{([tupl\rangle,n)}$ is given by the following graph* $\texttt{Term}_n$



*and the* $([prod], n)$*-marked diagram* $in_n^{Term} : \mathtt{Span}_n \hookrightarrow \mathtt{Term}_n$*. Finally,* $\mathcal{R}_{([tupl\rangle,n)}$ *is given by the graph* $\mathtt{Tupl}_n$*, the* $([tupl], n)$*-marked diagram* $id_{\mathtt{Tupl}_n}$*, the* $([prod], n)$*-marked diagram* $in_n^{Tupl} : \mathtt{Span}_n \hookrightarrow \mathtt{Tupl}_n$*, and* $n$ $[comp]$*-marked diagrams* $\delta_i : \mathtt{Triangle} \to \mathtt{Tupl}_n$*,* $i = 1, \ldots, n$ *with* $\delta_{i,0}(x) = y$*,* $\delta_{i,0}(y) = x$*,* $\delta_{i,0}(z) = x_i$*,* $\delta_{i,1}(1) = k$*,* $\delta_{i,1}(2) = p_i$*, and* $\delta_{i,1}(3) = r_i$*.*

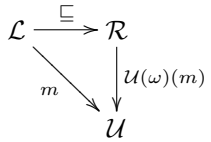
**Remark 11** (Construction of Names)**.** *Example 11 sheds some light on a very important* ***methodological point****: For many-sorted algebraic signatures we have the following induction rule for the construction of* $\Sigma$*-terms: For all* $op : s_1 \ldots s_n \to s$ *in* $OP$

$$t_1 \in T(\Sigma, X)_{s_1}, \ldots, t_n \in T(\Sigma, X)_{s_n} \quad \Rightarrow \quad op(t_1, \ldots, t_n) \in T(\Sigma, X)_s.$$

*In view of Generalized Sketches this means that we construct out of* $n$ *arrows* $X \to s_i$ *with "names"* $t_i$ *and one arrow* $s_1 \ldots s_n \to s$ *with "name"* $op$ *a new arrow* $X \to s$ *with the "compound name"* $op(t_1, \ldots, t_n)$*. That is, traditional, "syntactic" oriented approaches to formal specifications, as algebraic specifications, are mainly concerned about the construction of denotations (names) for new items out of the denotations (names) of given items. The fact that also new "arrows" are constructed is reflected, if at all, by the "typing" of the new items. In contrast, the Generalized Sketch framework is focusing on the construction of new nodes and new arrows. There is no build-in mechanism to construct names for the new items. We are convinced that a practical useful specification formalism has to combine both mechanisms. In Example 11 we could, for example, equip the* $\Theta_{\mathbf{EQ}}$*-sketch operations* $([tupl\rangle, n)$ *with a mechanism that creates for each match* $a : \mathcal{L}_{([tupl\rangle,n)} \to \mathcal{S}$ *the new denotation* $\langle t_1, \ldots, t_n\rangle$ *for the arrow* $a^*(k)$ *if* $t_i$ *is the denotation for* $a(r_i)$*,* $i = 1, \ldots, n$*.*

The next thing we may want to have, in an application, is that any model of $\mathcal{S}$ in a semantic universe $\mathcal{U}$ can be extended, in a unique way, to a model of $\omega(a)$ in $\mathcal{U}$. One possibility to gain this is to have a fixed interpretation of the sketch operation in the semantic universe $\mathcal{U}$.

**Definition 13** (Semantic Sketch Operations)**.** *For a* $\Theta$*-sketch operation* $\omega : \mathcal{L} \hookrightarrow \mathcal{R}$ *a* ***semantic*** $\Theta$***-sketch operation*** $\mathcal{U}(\omega)$ *in a semantic* $\Theta$*-universe* $\mathcal{U} = (\mathsf{U}, \mathcal{U}(\Pi), \mathsf{U}^m)$ *assigns to any* $\mathcal{L}$*-model* $m : \mathcal{L} \to \mathcal{U}$ *a* $\mathcal{R}$*-model* $\mathcal{U}(\omega)(m) : \mathcal{R} \to \mathcal{U}$ *such that* $Mod(\omega)(\mathcal{U}(\omega)(m)) = m$*, i.e., such that the following diagram commutes*




**Example 12** (Semantic Sketch Operations)**.** *We can define a semantics for the sketch operation* $[comp\rangle$ *in any semantic universe* $\mathcal{U}$ *just be the composition in the underlying category* $\mathsf{U}$*, i.e., for any* $\mathcal{L}_{[comp\rangle}$*-model* $m : \mathcal{L}_{[comp\rangle} \to \mathcal{U}$ *we can define* $\mathcal{U}([comp\rangle)(m)(3) = m(1); m(2)$*.*

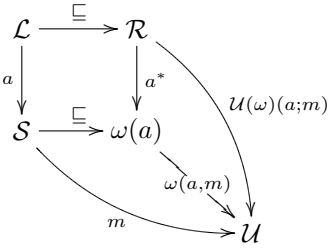
*Similar we can define a semantics for the sketch operation* $([prod\rangle, n)$ *in any semantic universe* $\mathcal{U}$ *with* $\mathsf{U}$ *equals to* Set*,* Par*, or* Pow *by the cartesian product of sets, i.e., for any* $\mathcal{L}_{([prod\rangle,n)}$*-model* $m : \mathcal{L}_{([prod\rangle,n)} \to \mathcal{U}$ *we can define* $\mathcal{U}([prod\rangle, n)(m)(x) = m(x_1) \times \cdots \times m(x_n)$*.*

*Note, that we have to make a choice between all the isomorphic possibilities to define products. Note further, that the cartesian product of sets is* ***not*** *the categorical product in* Par *or* Pow*, respectively.*

**Remark 12** (Partial Semantic Sketch Operations)**.** *The sketch operation* $[id\rangle$ *as defined in Example 11 will only give rise to a* ***partial semantic sketch operation*** *in a given semantic universe* $\mathcal{U} = (\mathsf{U}, \mathcal{U}(\Pi), \mathsf{U}^m)$*. To repair this we can define a new predicate label* $[id - node]$ *with arity the discrete graph* $(x\ y)$ *and with* $\delta : (x\ y) \to \mathsf{U}$ *in* $\mathcal{U}([id - node])$ *iff* $\delta_0(x) = \delta_0(y)$*. If we extend then* $\mathcal{L}_{[id\rangle}$ *by the* $[id - node]$*-marked diagram* $id_{(x\ y)}$ *and* $\mathcal{R}_{[id\rangle}$ *by the corresponding* $[id - node]$*-marked diagram* $in : (x\ y) \to \mathtt{Arrow}$*, we can trivially define a* ***total*** *semantic sketch operation* $\mathcal{U}([id\rangle)$ *by chosing* $\mathcal{U}([id\rangle)(m)(1) = id_{m(x)}$ *for any* $\mathcal{L}_{[id\rangle}$*-model* $m$*, i.e., for any sketch morphism* $m : \mathcal{L}_{[id\rangle} \to \mathcal{U}$ *with* $m(x) = m(y)$*.*

If we have such a semantic $\Theta$-sketch operation $\mathcal{U}(\omega)$ available we can extend now any $\mathcal{S}$-model $m : \mathcal{S} \to \mathcal{U}$ to an $\omega(a)$-model $\omega(a, m) : \omega(a) \to \mathcal{U}$ since $\omega(a)$ is defined by a pushout construction. That is, $\omega(a, m)$ is the

unique mediating morphism from $\omega(a)$ to $\mathcal{U}$ such that $\sqsubseteq;\omega(a,m) = m$ and $a^*;\omega(a,m) = \mathcal{U}(\omega)(a;m)$.




# 6 Dependencies

Dependencies between predicates are important for the theory and for the applications of Generalized Sketches. Therefore, we want to touch on this topic in this last section. We can identify, at least, two kinds of dependencies between predicates:

- Firstly, we have dependencies between single predicates as, for example, the dependency that the tupling of terms requires the existence a corresponding product. Those dependencies, that can be expressed by graph homomorphisms between the arities of predicates, may be called **requirements** and will be discussed in this section.
- Secondly, we have more complex (logical) dependencies if the satisfaction of a certain set of predicates entails the satisfaction of (sets of) other predicates. In UML, for example, the predicates [inv] for $f$, $g$ and [total] for $f$ entail the predicate [cover] for $g$. This kind of dependencies will be a topic of future research.

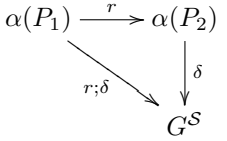
**Definition 14** (Requirements)**.** *Given a diagram signature $\Theta = (\Pi, \alpha)$, a graph homomorphism $r : \alpha(P_1) \to \alpha(P_2)$ with $P_1, P_2 \in \Pi$ is called a $\Theta$-**requirement** for $P_2$.*

**Remark 13** (Requirements)**.** *If $\alpha(P_1), \alpha(P_2)$ are discrete graphs, i.e., graphs $P_1 = (x_1 \dots x_m)$, $P_2 = (y_1 \dots y_n)$ without arrows, then a requirement $r : \alpha(P_1) \to \alpha(P_2)$ can be interpreted, in logical terms, as an implication: $P_2(y_1, \dots, y_n) \Rightarrow P_1(r(x_1), \dots, r(x_m))$. Note, that some variables $y_i$ may not appear in the conclusion, if $r$ is not surjective, and that some variables $y_i$ may appear at different places in the conclusion, if $r$ is not injective.*

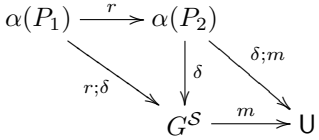
We can pose now two questions: What does it mean that a requirement is valid in a sketch, and when does a model satisfy a requirement?

**Definition 15** (Validity, Satisfaction)**.** *A $\Theta$-requirement $r : \alpha(P_1) \to \alpha(P_2)$ is **valid** in a $\Theta$-sketch $\mathcal{S} = (G^{\mathcal{S}}, \mathcal{S}(\Pi))$, $\mathcal{S} \Vdash_\Theta r$ in symbols, iff for all $P_2$-marked diagrams:*

$$(P_2, \delta : \alpha(P_2) \to G^{\mathcal{S}}) \in \mathcal{S}(P_2) \quad \Rightarrow \quad (P_1, r; \delta : \alpha(P_1) \to G^{\mathcal{S}}) \in \mathcal{S}(P_1).$$




*A $\Theta$-sketch model $m : \mathcal{S} \to \mathcal{U}$ of a $\Theta$-sketch $\mathcal{S}$ in a semantic $\Theta$-universe $\mathcal{U}$ **satisfies** a $\Theta$-requirement $r : \alpha(P_1) \to \alpha(P_2)$, $m \models_\Theta r$ in symbols, iff $(P_1, r; \delta; m : \alpha(P_1) \to \mathsf{U})) \in \mathcal{U}(P_1)$ for all $P_2$-marked diagrams $(P_2, \delta : \alpha(P_2) \to G^{\mathcal{S}}) \in \mathcal{S}(P_2)$.*




The satisfaction of requirements can be semantically ensured by the choice of an appropriate semantic universe.

**Proposition 7** (Satisfaction Semantically)**.** *For any $\Theta$-requirement $r : \alpha(P_1) \to \alpha(P_2)$, any $\Theta$-sketch $\mathcal{S}$, any semantic $\Theta$-universe $\mathcal{U}$, and any $\Theta$-sketch model $m : \mathcal{S} \to \mathcal{U}$ of $\mathcal{S}$ in $\mathcal{U}$ we have:*

$$\mathcal{U} \Vdash_\Theta r \quad \Rightarrow \quad m \models_\Theta r\,.$$

*Proof.* For any $P_2$-marked diagram $(P_2, \delta : \alpha(P_2) \to G^{\mathcal{S}}) \in \mathcal{S}(P_2)$ we have $(P_2, \delta; m) \in \mathcal{U}(P_2)$ since $m$ is a $\Theta$-sketch model. But, this implies $(P_1, r; \delta; m) \in \mathcal{U}(P_1)$ since $\mathcal{U} \Vdash_\Theta r$. □

On the other side, satisfaction can be syntactically enforced.

**Proposition 8** (Satisfaction Syntactically)**.** *For any $\Theta$-requirement $r : \alpha(P_1) \to \alpha(P_2)$, any $\Theta$-sketch $\mathcal{S}$, any semantic $\Theta$-universe $\mathcal{U}$, and any $\Theta$-sketch model $m : \mathcal{S} \to \mathcal{U}$ of $\mathcal{S}$ in $\mathcal{U}$ we have:*

$$\mathcal{S} \Vdash_\Theta r \quad \Rightarrow \quad m \models_\Theta r\,.$$

*Proof.* For any $P_2$-marked diagram $(P_2, \delta : \alpha(P_2) \to G^{\mathcal{S}}) \in \mathcal{S}(P_2)$ we have $(P_1, r; \delta) \in \mathcal{S}(P_1)$ since $\mathcal{S} \Vdash_\Theta r$. But, this implies $(P_1, r; \delta; m) \in \mathcal{U}(P_1)$ since $m$ is a $\Theta$-sketch model. □

If we want, we can now incorporate requirements into our concept of signature thus we will be allowed to write only specifications respecting these requirements.

**Definition 16** (Requirement Signature)**.** *A **requirement signature** $\mathbf{\Theta} = (\mathbf{\Pi}, \boldsymbol{\alpha})$ is given by*

- *a graph $\mathbf{\Pi}$ with nodes called **(predicate) labels** and with arrows called **(predicate) requirements***
- *a graph homomorphism $\boldsymbol{\alpha} : \mathbf{\Pi} \to \mathsf{Graph}$ assigning to each label $P \in \mathbf{\Pi}_0$ its **arity (shape)** $\boldsymbol{\alpha}_0(P)$ and to each (predicate) requirement $(d : P_1 \to P_2) \in \mathbf{\Pi}_1$ a $\mathbf{\Theta}_0$-requirement $\boldsymbol{\alpha}_1(d) : \boldsymbol{\alpha}_0(P_1) \to \boldsymbol{\alpha}_0(P_2)$ where $\mathbf{\Theta}_0 = (\mathbf{\Pi}_0, \boldsymbol{\alpha}_0)$ is the diagram signature part of $\mathbf{\Theta}$.*

**Definition 17** (Requirement Sketch)**.** *Given a requirement signature $\mathbf{\Theta} = (\mathbf{\Pi}, \boldsymbol{\alpha})$ a **$\mathbf{\Theta}$-sketch** $\mathcal{S}$ is a $\mathbf{\Theta}_0$-sketch such that $\mathcal{S} \Vdash_\Theta \boldsymbol{\alpha}_1(d)$ for all $d \in \mathbf{\Pi}_1$.*

**Remark 14** (Requirements in Tools)**.** *The use of requirement signatures $\mathbf{\Theta}$ in a tool for writing or drawing sketches can be chosen to be restrictive or supportive: Restrictive means that we can only add a new $P_2$-marked diagram $(P_2, \delta)$ after we have added before all the necessary $P_1$-marked diagrams $(P_1, \boldsymbol{\alpha}_1(d); \delta)$. Supportive, however, would mean that adding a new $P_2$-marked diagram $(P_2, \delta)$ generates also all $P_1$-marked diagram $(P_1, \boldsymbol{\alpha}_1(d); \delta)$ for all requirements $(d : P_1 \to P_2) \in \mathbf{\Pi}$.*

*In the same way requirement signatures may also help in visualising sketches since a visualization of a $P_2$-marked diagram $(P_2, \delta)$ allows to drop the visualization of the depending $P_1$-marked diagrams $(P_1, \boldsymbol{\alpha}_1(d); \delta)$.*

Proposition 8 ensures that any $\mathbf{\Theta}_0$-sketch $\mathcal{S}$-model $m : \mathcal{S} \to \mathcal{U}$ satisfies all the $\mathbf{\Theta}_0$-requirements $\boldsymbol{\alpha}_1(d)$ in $\mathcal{S}$, i.e., $m$ becomes is indeed a model of the whole $\mathbf{\Theta}$-sketch $\mathcal{S}$. This indicates that there should be no principal problems to extend the concepts, constructions and results from section 4 (and hopefully also from section 5) to requirement signatures. For example, it is obvious how to extend the concept of signature morphism since the step from "diagram signatures" to "requirement signatures" is simply modeled as a step from sets (of nodes) to graphs,

**Definition 18** (Requirement Signature Morphism)**.** *A **requirement signature morphism** $\varphi : \mathbf{\Theta}_1 \to \mathbf{\Theta}_2$ between two requirement signatures $\mathbf{\Theta}_1 = (\mathbf{\Pi}_1, \boldsymbol{\alpha}_1)$ and $\mathbf{\Theta}_2 = (\mathbf{\Pi}_2, \boldsymbol{\alpha}_2)$ is a graph homomorphism $\varphi : \mathbf{\Pi}_1 \to \mathbf{\Pi}_2$ such that $\boldsymbol{\alpha}_{2,0}(\varphi_0(P_1)) = \boldsymbol{\alpha}_{1,0}(P_1)$ for each $P_1 \in \mathbf{\Pi}_{1,0}$ and $\boldsymbol{\alpha}_{2,1}(\varphi_1(d_1)) = \boldsymbol{\alpha}_{1,1}(d_1)$ for each $d_1 \in \mathbf{\Pi}_{1,1}$.*

A detailed exposition of all these extensions has to be left for a forthcoming paper.

# 7 Historical remarks, relation to other and future work

Applications of categorical logic to data modeling were first described, probably, in [DJM92], with a major emphasis on commutative diagrams and less on the universal properties. In the same context of data modeling, machinery of generalized sketches was developed and applied in a few industrial projects in Latvia in 1993-94, and the corresponding logic presented at Logic Colloquium'95 [Dis97b]. Even earlier, Michael Makkai came to the need to generalize the notion of Ehresmann's sketches from his work on an abstract formulation of Completeness Theorems in logic. Makkai attributed the immediate impetus for him to work out these ideas to Charles Well's talk at

the Montreal Category Theory Meeting in 1992. Well's own work went in a somewhat different direction [BW97] while Makkai's work resulted in the notion (and the very term) of generalized sketch. Makkai also developed a corresponding mathematical theory that was first presented in his several preprints circulated in 1993-94, and summarized and published later in [Mak97].

Relations between generalized sketches as they are understood by Makkai and other generalization of the Ehresmann's concept are discussed in the introduction to [Mak97]. Formally speaking, the *"Latvian"* version of the definition coincides with (or perhaps is slightly less general than) Makkai's definition. The difference is that the Latvian version stresses the parallelism between syntactical structures of ordinary FOL specifications and generalized sketches as much as possible: the former can be seen as a logic of sketches over the base category Set while the latter is a logic over Graph, the category of graphs. The most detailed presentation of the Latvian interpretation is in [Dis97a]; unfortunately, it also involves a misleading attempt to include into the notion of generalized sketch some issues related to visualization of sketches rather than to the logic as such.

**Future work.** A well-known observation is that state transformations can **not** be described by model morphisms. One possible way to describe those transformations is by spans of model morphisms [JR01]. A detailed investigation of different possibilities to model state transformations is needed.

The parallel and sequential composition of sketch operations has to be investigated and an appropriate concept of "sketch term" and of "term substitution" has to be developed.

The whole topic of "sketch logic" has not been discussed in the paper. We have to address here logical connectives and corresponding "derived predicate labels", and a proper notion of substitution has to be defined. An appropriate extension of the concept of signature morphism that would allow us to map predicate labels to "derived predicate labels" will be also necessary in application domains. Finally, it will be interesting to study the relation between Generalized Sketches and abstract concepts of logics as described in the Institution framework [GB92].

# 8 Conclusions

We have given a new restructured presentation of the basic concepts and results of the Generalized Sketch Framework. We made especially apparent that Generalized Sketches appear as a natural generalization of FOL specifications, where there are "node" and "arrow" variables organized into a graph. Generalized Sketches offer a universal format for defining syntax and semantics of diagrammatic specification techniques and for describing the transitions between those techniques as well. This can be seen in analogy to the BNF that provides a universal format for defining syntax of programming languages and transitions between them.

The sketch formalism appears to be a good design pattern for diagrammatic language design. It suggests that if one is thinking about designing a new language, one should address the following three basic questions: what is the interpretation of nodes, what is it for arrows, and what is the signature of diagram predicates that matter. Brevity, clear semantics and solid mathematical foundations can make this pattern really helpful in practice. Moreover, active promotion of modeling techniques into software industry and, particularly, the rapid progress of the *domain-specific languages* sector (so called DSL), form an explicit industrial demand for convenient and handy yet reliable language design techniques. We believe that the generalized sketch pattern provides a promising theoretical supply to the demand.

It would not be a big exaggeration to say that modeling languages are designed daily in the modern software industry. Hopefully the next one hundred modeling languages will be designed along the lines of the generalized sketch pattern.

# References


[BW95] M. Barr and C. Wells. *Category Theory for Computing Science*. Prentice Hall International Series in Computer Science, 1995. 1, 1, 5, 4, 4

[BW97] A. Bagchi and C. Wells. Graph-based logic and sketches. In *10th Int.Congress of Logic,Methodology and Philosophy of Science, Florence, 1995*, Florence (Italy), 1997. Kluwer Acad.Publ. 7

[DD06] Z. Diskin and J. Dingel. Mappings, maps and tables: Towards formal semantics for associations in UML2. In *Proc. ACM/IEEE 9th Int.Conference on Model Driven Engineering Languages and Systems*, 2006. 8

[Dis97a] Z. Diskin. Generalized sketches as an algebraic graph-based framework for semantic modeling and database design. Technical Report M9701, Faculty of Physics and Mathematics, University of Latvia, 1997. 33pp., http://www.cs.queensu.ca/ zdiskin/Pubs/ULReport-M97.pdf. 7

[Dis97b] Z. Diskin. Towards algebraic graph-based model theory for computer science. *Bulletin of Symbolic Logic*, 3:144–145, 1997. Presented (by title) at *Logic Colloquium'95*. 5, 7

[Dis02] Z. Diskin. Visualization vs. specification in diagrammatic notations: A case study with the UML. In *Diagrams'2002: 2nd Int. Conf. on the Theory and Applications of Diagrams*, Springer LNAI#2317, pages 112–115, 2002. 1

[Dis03] Z. Diskin. Mathematics of UML: Making the Odysseys of UML less dramatic. In Ken Baclawski and Haim Kilov, editors, *Practical foundations of business system specifications*, pages 145–178. Kluwer Academic Publishers, 2003. 1, 2

[Dis05] Z. Diskin. Mathematics of generic specifications for model management. In Rivero, Doorn, and Ferraggine, editors, *Encyclopedia of Database Technologies and Applications*, pages 351–366. Idea Group, 2005. 5

[DJM92] 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. 7

[DK97] Z. Diskin and B. Kadish. A graphical yet formalized framework for specifying view systems. In *Advances in Databases and Information Systems*, pages 123–132, 1997. ACM SIGMOD Digital Anthology: vol.2(5), ADBIS'97. 5

[DK03] Z. Diskin and B. Kadish. Variable set semantics for keyed generalized sketches: Formal semantics for object identity and abstract syntax for conceptual modeling. *Data & Knowledge Engineering*, 47:1–59, 2003. 1, 7, 4

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformations*. EATCS Monographs on Theoretical Computer Science. Springer, Berlin, 2006. 10

[FB05] F. Fondement and T. Baar. Making metamodels aware of concrete syntax. In *ECMDA-FA*, pages 190–204, 2005. 1

[Fia05] J. L. Fiadeiro. *Categories for Software Engineering*. Springer, Berlin, 2005. 1, 2, 4

[GB92] J. A. Goguen and R. M. Burstall. Institutions: Abstract Model Theory for Specification and Programming. *Journals of the ACM*, 39(1):95–146, January 1992. 7

[Gog98] J. Goguen. An introduction to algebraic semiotics, with applications to user interface design. In C. Nehaniv, editor, *Computation for Metaphors, Analogy and Agents*, pages II: 54–79. Univertsity of Aizu, 1998. The latest version is available at http://ww-cse.ucsd.edu/users/goguen. 1

[Hec06] R. Heckel. Graph Transformation in a Nutshell. In *Proceedings of the School on Foundations of Visual Modelling Techniques (FoVMT 2004) of the SegraVis Research Training Network*, volume 148 of ENTCS, pages 187–198. Elsevier, 2006. 10

[JR01] M. Johnson and R. Rosebrugh. View updatability based on the models of a formal specification. In *FME*, pages 534–549, 2001. 10, 7

[JRW02] M. Johnson, R. Rosebrugh, and R. Wood. Entity-relationship-attribute designs and sketches. *Theory and Applications of Categories*, 10(3):94–112, 2002. 1, 1

[LS86] J. Lambek and P. Scott. *Introduction to higher order categorical logic*. Cambridge University Press, 1986. 6

[Mak97] M. Makkai. Generalized sketches as a framework for completeness theorems. *Journal of Pure and Applied Algebra*, 115:49–79, 179–212, 214–274, 1997. 5, 7

[PS97] F. Piessens and E. Steegmans. Proving semantical equivalence of data specifications. *J. Pure and Applied Algebra*, (116):291–322, 1997. 1

[Rei87] H. Reichel. *Initial Computability, Algebraic Specifications, and Partial Algebras*. Oxford University Press, 1987. 9, 8

[RJB04] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual. Second Edition.* Addison-Wesley, 2004. 1

[Sel06] Bran Selic. Model-driven development: Its essence and opportunities. In *ISORC*, pages 313–319, 2006. 1

[Wel] C. Wells. Sketches: Outline with references. Available under http://www.cwru.edu/artsci/math/wells/pub/papers.html. 1

[WKWC94] U. Wolter, M. Klar, R. Wessäly, and F. Cornelius. Four Institutions – A Unified Presentation of Logical Systems for Specification. Technical Report Bericht-Nr. 94-24, TU Berlin, Fachbereich Informatik, 1994. 9

[Wol90] U. Wolter. An Algebraic Approach to Deduction in Equational Partial Horn Theories. *J. Inf. Process. Cybern. EIK*, 27(2):85–128, 1990. 9, 8