Sketches, Queries, and Views: From Term-Graphs to Diagrammatic Term-Sketches

Revision of TR 2020-33

Zinovy Diskin and Patrick Stünkel
Sketches, Queries, and Views: From Term-Graphs to Diagrammatic Term-Sketches

Revision of TR 2020-33

Zinovy Diskin\textsuperscript{1} and Patrick Stünkel\textsuperscript{21}

\textsuperscript{1} McMaster University, Centre for Software Certification, Hamilton, Canada
diskinz@mcmaster.ca
\textsuperscript{2} Høgskulen på Vestlandet, Institutt for Datateknologi, Elektroteknologi og Realfag, Bergen, Norway past@hvl.no

Abstract. A diagrammatic operation takes a diagram of a specified shape as its input and completes it with new elements forming a bigger diagram (e.g., arrow composition takes two consecutive arrows as its input and adds a third arrow resulting in a triangle). Pre- and post-conditions are an important part of the story (e.g., monic preservation by pullbacks) so that diagram chasing smoothly integrates logic and algebra. This technique transcends category theory and finds numerous applications in practice of software engineering, specifically in the multi-view approach to system design, and in model management (in the sense of model-driven engineering). These applications force us to consider a very general version of diagram chasing over generalized sketches (in the sense of Makkai) and give rise to a project (now in progress) reported in the present technical Report.

We introduce the notions of diagrammatic signatures, terms and algebras over an underlying sketch category, and prove several basic results about them; specifically, that the category of views is symmetric monoidal.

Our views are a modification of the standard notion of a Kleisli mapping, in which each view $v : V \rightarrow Q(v)(S)$ carries its own term $Q(v)$ for extending the target $S$ of the mapping rather than using one unified but huge monad. Nevertheless, we prove that our localized Kleisli mappings generate a classical Kleisli triple and hence a monad.
1 Introduction

A diagrammatic operation takes a diagram of a specified shape as its input, and
extends it with new elements forming a bigger diagram of a specified shape;
we will say that the input diagram is completed by the operation. For example,
arrow composition takes two consecutive arrows and completes them with a
third arrow forming the corresponding triangle. The resulting diagram often
satisfies some postconditions, e.g., a chosen pullback completes a cospan with
a commutative square, and a chosen powerobject completes an object with its
jointly-monic membership span. Preconditions are needed too, e.g., adhesive
categories only provide pushouts for one-leg-monic spans (and then we know that
1 Introduction

the corresponding leg of the resulting coequi is also monic). Thus, categorical
diagram operations come together with reasoning over them so that diagram
algebra and logic are smoothly integrated in diagram chasing routines.

This paper stems from the observation that the diagram chasing idea tran-
scends Category Theory (CT) and finds numerous applications in the practice of
Software Engineering (SE), in which, however, it is currently used in an ad-hoc
way: implicit, semi-formal or formal but buried in code so that the underly-
ing diagram chasing procedures are blurred and distorted. The negative impact
of these issues increases with the recent advent of Model Driven Engineering
(MDE), in which diagrammatic models rather than code become the primary
artifacts of software development.

Our goal in the project reported in the present technical report (further TR)
is to provide an accurate formalization of diagrammatic reasoning routines ade-
quate to their practical use in MDE. We will briefly discuss this in the next
section (Sect. 2.1-2) along with several mathematically interesting structures
and results emerged along the course (in Sect. 2.3); a brief survey of related
work and our contributions are in Sect. 2.4. We also describe a simple exam-
ple of domain modelling motivating generalized sketches as the main underly-
ing machinery in Sect. 2.5, and overview our notation in Sect. 2.6. Section 3 gives
three brief introductions to our three main subjects: diagrammatic logic, dia-
grammatic algebra, and diagrammatic reasoning. Sections 4-6 present technical
development: in Sect. 4 we consider sketches over predicate signatures (with or
without dependencies between predicates), and in Sections 5-6 complex queries
built from diagrammatic operations (the case of operation without dependencies
is in Sect. 5 and the general case with dependencies in Sect. 6; the latter is is in
progress and currently only provides basic definitions). Section 7 concludes.
2 Background and Contributions

We discuss two main applications of the framework in SE in Sect.2.1-2 (plus Sect. A in Appendix), then discuss some mathematical aspects of the framework in Sect.2.3, and describe related work and contributions in Sect.2.4. Sect. 2.5 overviews our notation.

2.1 Views.

Developing a complex software system normally requires working with multiple views [1] of the system: This is the only way to manage complexity and a cornerstone of system engineering, see [18] for an early discussion and [17,5,6] for more recent reports and surveys. A view of a source artifact $S$ is, in general, not merely a part of $S$ or a projection of $S$, but typically involves some computation over $S$, e.g. composing references, aggregating attributes by summing up their values etc, which extends $S$ with derived elements produced by some predefined operations (we will say queries) applied to $S$. Then a view can be defined as a mapping $v: V \rightarrow Q_v(S)$ in a suitable category of software artifacts where the source $S$ and the view schema $V$ live, and $Q_v$ is a query that provides an extension $Q_v(S)$ of source $S$ with derived elements required by the view. Section A (in Appendix) presents an example.

Note an important distinction from the ordinary Kleisli mappings defined for a given monad $Q$ or, equivalently, for a global mapping on objects, which only potentially exists and seems unnecessary abstract for software engineers not familiar with Haskell. In contrast, in our notion of view, each view mapping $v$ carries its own query $Q_v$ and maps to its own local extension $Q_v(S)$ rather than to a unified but huge global extension $Q(S)$. A simple example in Sect. A shows how it works, and also demonstrates the importance of constraints for the view mechanisms. It is placed aside to save time for the first reading, but we encourage the reader to take a look there to obtain a notion of the MDE world and our motivation for this paper.

2.2 Algebra for model management.

In model management, models are manipulated as holistic entities, and replacement the one-object-at-a-time approach (micromanagement) by one-model-at-a-time (macromanagement) turned out very productive [2,16,26]. It requires considering high-level operations on models such as model slicing, transformation, merging, and synchronization, and composing them into complex workflows [11]. Fig. 1 shows a simple example in which two views of the same source are first matched, and then merged. One may think of this as taking the pullback and then the pushout, but in practice these “pure” categorical operations are often interwoven with some normalization procedures (and may require conflict resolution) so that even in this simple case the real workflow will be much richer. Importantly, models in SE are typically considered up to their auto-isomorphisms generated by permutations of OIDs (in the sense of OO programming) which are irrelevant.
and normally not visible. It justifies making categories of models skeletal, which
makes limits and colimits deterministic algebraic operations. Thus, for both the
inter-model and the intra-model levels of reasoning, SE needs powerful and for-
manly specified algebraic machinery for diagrammatic operations. On the other
hand, it seems that even basic universal algebraic constructs (signature, term,
parsing) have not been accurately specified for diagrammatic operations, espe-
cially when the latter are considered together with pre-and post-conditions and
hence are interwoven with logic; in this sense, we are developing some basic
universal algebra for diagrammatic operations.

2.3 Mathematical aspects

Why generalized graphs and generalized sketches
As discussed above, categories hosting software models are categories of sketches; Sect. A (in Ap-
pendix) presents an example). In general, a sketch is a graphical structure en-
dowed with labelled diagrams (a.k.a. constraints); in the project, we consider
a very general version of the notion: our 'graphs' are finitary presheaves, and
our constraints of shape $G$ are any “devices” that classify mappings from $G$ as
either valid or invalid. The generality of our graphs is caused by the great di-
versity and complexity of graphical structures used in SE [21], and graphs used
for behavioural and workflow models can even be more complex. Similarly, con-
straints used in domain modelling (banking, e-commerce, automotive etc.) can
be complex and very elementwise-oriented so that their representation in a stan-
dard categorical way via universal properties can be rather bulky and unwieldy
(a detailed discussion can be found in [13], see also Appendix). Thus, we need
constraints and sketches of a very general nature; Michael Makkai called them
generalized sketches in [25].

Why sketches with diagrams operations rather than string diagrams
The objects in a $PROP$ are natural numbers, and arrows represent operations,
where domain and codomain denote in- and output arity (operations are multiary
but one-sorted). Being symmetric monoidal (sm) categories, $PROPs$ can be
visualized with String Diagrams [30], an intuitive yet formal graphical syntax.
For example, string diagrams in Fig. 2 depicts a ternary term $(x+y)\ast (x+z)$ built
from two binary operations. Note that reuse of the middle variable $y$ is shown
explicitly with Copy-operation (visualized via $\leftarrow$). This is possible as set $\{x, y, z\}$

\[\begin{array}{ccc}
x + y & \ast & (x+y)\ast (y+z) \\
y & \leftarrow & (y+z) \\
z & \leftarrow & (x+y)\ast (y+z)
\end{array}\]

Fig. 2: String diagram notation for ordinary algebraic terms

is the coproduct of its elements $\{x\} \sqcup \{y\} \sqcup \{z\}$. However, diagram operations
are graph-based, and a graph (while being a colimit of its elements) is not their
coproduct!
Sketch $X$ (black) enriched with derived elements (dashed blue)

a) Categorical notation with explicit labelling

b) Accurate formalization of a) in the PO-calculus

c) A monoidal counterpart of a) via a string diagram

Fig. 3: Diagram Chasing
Figure 3 presents a simple example showing how the machinery of sketches work. Diagram (a) is a typical example of categorical diagram chasing. Suppose we have a diagram operation \([pb]\) defined over any graph: it takes a cospan of arrows as its input and returns a span of arrows, which together with the input cospan forms a square in a familiar way. In addition, we assume there is a predicate symbol \([\text{monic}]\), which can be declared for an arrow. Moreover, if an input arrow for a \([pb]\)-application is monic, then the corresponding output arrow is monic too; this is usually termed as “monics are stable under \([pb]\)”. The left (black and blue) sub-diagram in (a) shows a diagrammatic term formed by three applications of \([pb]\).

We begin with a graph consisting of three arrows \(\{x, y, z\}\) with a common target \(O\). Applications \(a\) and \(b\) of \([pb]\) extend the graph with four arrows as shown, and two of them are monic according to the “monic are stable” property. (We paint the operation chevron and elements it produces into blue to recall the mechanical computation nature of these operations.) Then we apply \([pb]\) yet another time and produce two more monic arrows. In everyday categorical routines, the tentacle arrows are not needed because i) any user of categorical routines knows the arity shape of operation \([pb]\), and ii) the operation is commutative in the sense that \((y_1, y_2) = [pb](x_1, x_2)\) implies \((y_2, y_1) = [pb](x_2, x_1)\).

We will refer to the entire black-blue sketch as to a *term-sketch* or a query \(Q(X)\), which can roughly be considered as an application of query \(Q = (a + b); c\) to sketch \(X\). Note that queries extend a given input sketches so that we always have embedding \(\eta_Q: X \rightarrow Q(X)\).

Suppose now that for further processing, we only need a part of the entire black-blue sketch as shown by mapping \(v\) (in orange) that maps the interface sketch \(V\) to sketch \(Q(X)\); note that \(v: Q(X) \leftrightarrow V\) is a correct sketch morphism as it respects all diagrams in sketch \(V\) (in the example, \(V\) has just one diagram, but if it has more, they all are to be respected by mapping \(v\)). Thus, pair \((Q, v)\) makes a cospan \(X \xrightarrow{\eta_X} Q(X) \xleftarrow{v} V\), which can be seen as an arrow from \(X\) to \(V\). In different terminological disciplines, such a cospan could be called open term-sketch, or rooted term-sketch (and then sketch \(X\) would be referred to as a sketch of variables). We will refer to pair \((Q, v)\) as to a *view* of \(X\) with schema \(V\) and view (definition) mapping \(v\); or else as a *combinatorial Kleisli mapping*.

Although notation in diagram (a) is accurate, it is not entirely formal: the actual formal procedures underlying diagram (a) are shown in diagram (b). Arity of \([pb]\) is specified as a monic between two graphs, and the presence of the monic label in the input implies yet another monic label in the output. It is clear that in diagram a), input tentacle arrows into chevrons \(a;[pb], b;[pb]\) actually denote mappings \(\beta_a, \beta_b\), and together with output tentacles they denote mappings \(\beta'_a, \beta'_b\) into sketches produced by the corresponding pushouts \(PO_a, PO_b\) (arrows with stroked tails and hollow triangle heads (like for inheritance in the UML) denote monics). Note that we need yet another PO to produce the resulting sketch \(X_a +_X X_b\) formally. Then we can apply \([pb]\) yet another time by defining a mapping \(\beta_c\), and so on. Thus, notation in diagram a) hides a series of PO computations under the hood similarly to how string diagrams hide axioms of hypercategories.

In the report, we will show that sketches and views can be endowed with a structure of a *semi-hyper-category* (\(s\text{Hyp-category}\)), i.e., symmetric monoidal category with two additional operations, duplicator/copying and discharger (garbage collector), which satisfy the monoidality and coherence axioms. Such categories
under the name of gs-monoidal categories are discussed in [7] in detail. The tensor product is played by the (chosen) sketch coproduct, and the sequential composition is done via pushout, which is nothing but query substitution.

Diagram c) presents a set-based counterpart of the term a), in which operation \([pb]\) is simulated by operation \(*\) of arity \((2,2)\) (in fact, \((2,2)\) is an arity-coarity pair but we will call it arity): it takes two elements as its input and outputs two elements denoted by \(_*1_\) and \(_*2_\). Sharing of variable \(y\) can be modelled by applying operation \(\text{copy}\) of arity \((1,2)\), and exclusion of elements from the view is modelled by applying discharger operation of arity \((1,0)\). The string diagram c) can thus be seen as a term in the signature of semi-hyper-categories. In contrast, diagrammatic term a) is not an sHyp-term: although arrow \(y\) is shared, we cannot model it by copying due to the graphical structure: while a set is a tensor (coproduct) of its elements, a graph is not. Similarly, we cannot model exclusion of arrows \(y_x\) and \(y_z\), and label \(c_1: [\text{monic}]\) from the view by applying an “arrow discharger” or a “label discharger” due to structural constraints imposed on those elements. The structure of sH-category over diagrammatic term-sketches only appears on the macro level, when we consider ports whose types are sketches rather than elements.

### 2.4 Related Work and Contributions

By *logic* we refer to work on generalized sketches, and *algebra* refers to work on signatures of operation symbols, terms and algebras over them.

**Logic.** Several authors noticed that Ehresmann’s sketches can be applied to data modelling [23,27,20]. However, these sketches are based on universal properties of the corresponding diagrams (limit, colimit, powerobject), which (as we discussed above) is a too restrictive setting for applications. A suitable generalization of sketches motivated by exactly this reason was proposed by Diskin [8,9], but even earlier, in a very different context of an abstract approach to logic, the same generalization was proposed by Makkai, whose preprints had been circulating in early 90s but a summarizing paper [25] was published later in 1997. Makkai developed a coherent theory of generalized sketches and proved a fundamental result that any category of sketches is a presheaf topos. Hence, sketches in a signature \(P_1\) can be considered as graphs, and hence a new signature \(P_2\) and a new category of \(P_2\)-sketches over \(P_1\)-sketches-as-graphs can be defined, and so on. In this way, Makkai specified categories with several layers of additional structure as sketches. In our project, we use a different approach in which all discrete signatures \(P_i\) are integrated into one signature-category \(\int P_i\), whose arrows describe dependencies between predicate symbols — an idea first proposed by Wolter and employed in [13]. We prove that Makkai’s theorem can be extended for categorical signatures so that the slogan “sketches are graph” persists (see Th. 1 for a precise formulation). Also, while traditionally sketches are defined in an indexed way as families of sets of diagrams indexed by predicate symbols (for the discrete signature case), or functors from the signature (for the categorical signature case [13]), we define them in an equivalent fibrational way via a discrete opfibration functor into the signature: the fibrational definition is more compact and often simplifies technicalities.

To give a detailed overview of what is done in the paper in comparison with other work, we need to distinguish between several notions of signatures and
sketches. We call a logic basic if signatures of predicate symbols are sets considered as discrete categories. We call it advanced, if signatures are categories whose arrows are dependencies between predicates; in addition, these arrows are required to go in one direction, which makes signature categories hierarchical (a version of Lawvere’s one-way categories). There is also an important distinction between multi-sketches that allow the same labelled diagram appear in a sketch for multiple times, and mono-sketches that disallow this. Finally, a partially decategorificated approach (due to Makkai) is to consider signatures as layered sets so that each predicate in layer \((n+1)\) implicitly depends on all predicates in layer \(n\). However, when arities of \((n+1)\)-predicates are defined based on \(n\)-predicates, the actual dependencies appear on the stage. We call this approach discrete hierarchical (dh) and show that it is actually equivalent to the categorical one. In these terms, Table 1 shows the previous work and contributions of the project. Our main results here are an extension of a fundamental Makkai’s theorem (that multi-sketches are actually presheaves) for the case of categorical signatures with dependencies, Th. 1, and an equivalence of the two approaches to sketches: internally categorical (signatures are categories) and discrete hierarchical (a hierarchy of discrete signatures), Th. 2.

<table>
<thead>
<tr>
<th>Sketch Def.</th>
<th>Setting</th>
<th>One-Layer (Basic Logic)</th>
<th>Multi-Layer (Advanced Logic)</th>
<th>Discrete Hierarchical</th>
<th>Categorical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fibrational</td>
<td>multi-sketches</td>
<td>Sect. 3 (Motivational)</td>
<td>Sect. 4.4</td>
<td>Sect. 4.2-4.3, Th. 1</td>
<td></td>
</tr>
<tr>
<td>(both mono &amp; multi)</td>
<td></td>
<td></td>
<td></td>
<td>Equivalence, Th. 2</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Diagram Logic and Contributions

**Algebra.** Diagram operation signatures and algebras were defined first in [9] and with some additional details in [8], but no mathematical results were obtained. For the special case of operations over ordinary graphs, Wolter et al [32] proved that any signature of such operations generates a monad (i.e., a global construct) over the category of graphs. The machinery of local Kleisli mappings explained informally was used in several papers of the first author, e.g., [14,10], and formalized as Kleisli mappings of a monad (i.e., globally) in [12]; we are not aware of an accurate formalization of Kleisli mappings considered locally as is done in the present TR, nor of several other results reported in Sect. 5. Specifically, we prove that a) sketches and local Kleisli mappings (views) between them form a symmetric monoidal category, and b) the local Kleisli composition gives rise to a Kleisli triple and hence a monad (as expected). Table 2 provides an overview of related work in this area and where to find our results in this paper.

### 2.5 Notation

We fix some of our notation and terminology. To manage size issues, we consider all categories in the paper are small and refer to the idea of Grothendieck uni-
verses for handling categories whose collections of objects and morphisms are “bigger” than sets. For variables referring to categories, we use blackboard font, e.g. \( \mathbb{C} \). Names of special categories such as \textbf{Set} or \textbf{Cat} are in bold italic.

**Notation for objects and arrows.** For a small category \( \mathbb{C} \), its set of objects and set of arrows (morphisms) will be denoted by \( \text{Ob}(\mathbb{C}) \) and \( \text{Arr}(\mathbb{C}) \) resp. We will also write \( \text{Elem}(\mathbb{C}) \) for the set \( \text{Ob}(\mathbb{C}) \sqcup \text{Arr}(\mathbb{C}) \).

As a rule, we will use uppercase letters for objects and lowercase letters for morphisms; then we can use this agreement to ease formulas and write \( f \) \( \in \text{C} \) meaning \( X \in \text{Ob}(\mathbb{C}) \) and \( f \in \mathbb{C} \) meaning \( f \in \text{Arr}(\mathbb{C}) \). The domain of an arrow \( f \) will be denoted by \( \text{dom}(f) \) or sometimes \( \circ f \), and the codomain by \( \text{cod}(f) \) or \( f \circ \) (one may as well think of \( \text{dom} \) and \( \text{cod} \) as mappings from \( \text{Arr}(\mathbb{C}) \) to \( \text{Ob}(\mathbb{C}) \)).

We write \( f: X \rightarrow Y \) (or \( X \xrightarrow{f} Y \) in diagrams) for an arrow \( f \in \text{Arr}(\mathbb{C}) \) with \( \text{dom}(f) = X \) and \( \text{cod}(f) = Y \).

The hom-set of all arrows between two objects \( X,Y \in \mathbb{C} \) will be denoted \( \mathbb{C}(X,Y) \). We write \( \mathbb{C}(X, \_ ) \) for \( \bigcup_{Y \in \text{Ob}(\mathbb{C})} \mathbb{C}(X,Y) \) and similarly for \( \mathbb{C}(\_, Y) \).

**Arrow composition and application.** Composition of two arrows \( f: X \rightarrow Y \) and \( g: Y \rightarrow Z \) will be denoted either in the diagrammatic order \( f \circ g: X \rightarrow Z \) or the application order \( g \circ f: X \rightarrow Z \).

For a mapping \( f: X \rightarrow B^A \) and \( x \in X \), we will often write \( f^x \) or \( f_x \) rather than \( f(x) \) to make reading applications of \( f(x) \) to elements of \( A \) easier: \( f^x(a) \) rather than \( f(x)(a) \) (although the latter form will be used too). Similarly, if \( f(x) \) is a domain of another arrow \( g: f(x) \rightarrow y \), we will often write \( x^f \) for \( f(x) \) so that \( g: x^f \rightarrow y \).

To highlight special properties of morphisms, we denote a monomorphism \( m \) by \( m: X \rightarrow Y \) and an epimorphism \( e \) by \( e: X \rightarrow Y \). Finally, we will interchangeably use expressions \( \mathbb{C}^\mathbb{D} \) or \( [\mathbb{C}, \mathbb{D}] \) to denote the functor category between \( \mathbb{C} \) and \( \mathbb{D} \), \( \mathbb{C}/X \) for the slice category under \( X \in \text{Ob}(\mathbb{C}) \), \( \mathbb{C}\backslash X \) for the coslice category over \( X \in \text{Ob}(\mathbb{C}) \), and \( \mathbb{C} \times \mathbb{D} \) for the product category. Class \( \text{Ob}([\mathbb{C}, \mathbb{D}]^*) \) will be often denoted by \([\mathbb{C}, \mathbb{D}]^* \) or just \([\mathbb{C}, \mathbb{D}] \) if the context makes discretization clear.

If \( f: \mathbb{C} \rightarrow \mathbb{D} \) is a functor and \( \mathbb{C}' \subseteq \mathbb{C} \) is a subcategory, then the restriction of \( f \) on \( \mathbb{C}' \) (i.e., composition \( i_{\mathbb{C}'} \circ f \) where \( i_{\mathbb{C}'} \) is the corresponding embedding) is denoted by \( \text{proj}_{\mathbb{C}'} \).

**Carriers.** In some contexts, where the domain of a mapping gives the name of the story, we will use the following (categorically, rather odd) naming for

<table>
<thead>
<tr>
<th>Setting</th>
<th>Discrete</th>
<th>Categorical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sect. 3.2; [32] (graph operations)</td>
<td>Sect. 5.1</td>
<td>Sect. 6.1, 6.2</td>
</tr>
<tr>
<td>Multiqueries and Substitution</td>
<td>Sect. 5.2</td>
<td></td>
</tr>
<tr>
<td>Query Semantics</td>
<td>Sect. 5.3</td>
<td></td>
</tr>
<tr>
<td>Complex Queries &amp; Composition</td>
<td>Sect. 5.5; [32] (graph term algebra adjunction)</td>
<td></td>
</tr>
</tbody>
</table>
components of an arrow \( f: X \rightarrow Y \). We will consider this arrow as a pair \( X = ([X], f_X) \) where the domain of \( f \) is now denoted by \([X]\) and called the \textit{carrier} of \( X \) while the arrow itself is denoted by \( f_X \) and is referred to as the \textit{mapping} of \( X \). All these are fully terminological permutations as \( f \) and \( X \) are the same formal construct. Moreover, by a common abuse of notation, we will overuse symbol \( X \) to refer to the carrier \([X]\), and will often omit the superscript \( X \) in \( f_X \).

3 Diagram Logic and Algebra: Overview

We will give three brief introductions: to a sketch version of diagrammatic predicate logic in Sect. 3.1, to a diagrammatic algebra in Sect. 3.2, and to diagrammatic reasoning in Sect. 3.3

3.1 Diagram Logic via Generalized Sketches: Basic Notions

A paradigmatic object of logical studies is a satisfiability relation between \textit{theories}—syntactical constructs built from predicate symbols, variables and logical symbols, and \textit{models}—semantic constructs built from predicate symbols, values and set-theoretical operations interpreting logical symbols. Categorical logic places syntax and semantics in a suitable category \( \mathbb{L}_P \) where \( P \) is a fixed signature and we will omit the subindex, and makes a model \( M \) of a theory \( T \in \text{Ob}(\mathbb{L}) \) an \( \mathbb{L} \)-morphism \([\cdot]_M: T \rightarrow [M]\) where \( [M] \) is an \( \mathbb{L} \)-object called the \textit{carrier} of model \( M \). In the sketch view of logic, the logical part is very simple: the only logical symbol is conjunction so that theories are merely sets of formulas, but the carrier part is interesting: carriers are graph-based structures that can be rather complex (2-graphs, hypergraphs, Petri nets, etc) and further are referred to as graphs. Thus, arities of predicate symbols are graphs, and a collection of variables \( X \) is a graph too, so that an atomic formula over \( X \) becomes a \textit{labelled} diagram \((P, d)\) with \( P \) a predicate symbol (of arity, say, \( \alpha(P) \)), and \( d: \alpha(P) \rightarrow X \) a diagram (graph morphism) in \( X \) of shape \( \alpha(P) \). A sketch is nothing but a collection of labelled diagrams (i.e., of diagrammatic atomic formulas). A suitable choice of a predicate signature can make this simple formalism extremely expressive. In this section, we will give simple variants of several basic definitions to show how the machinery works, and postpone a more detailed study until Sect.4-5.

\textbf{Generalized graphs.} Ordinary (directed, multi-)graphs can be seen as functors (presheaves) \( G: \text{Graph} \rightarrow \text{Set} \) over schema specified in (1)

\[
\begin{array}{ccc}
\text{Node} & \xrightarrow{\text{source}} & \text{Arrow} \\
\text{trg} & &
\end{array}
\]

(1)

Schema \text{Graph} for ordinary (directed multi-)graphs

Graph morphisms are natural transformations, and the category of all graphs is a presheaf topos \( \text{Set}^{\text{Graph}} \). Our applications require to consider more general notions of graphs that could be seen as presheaves over schemas \( G \) more complex than \text{Graph}. In general, a \textit{graph} is a presheaf \( G: G \rightarrow \text{Set} \) over a small category \( G \), and the category of graphs is a presheaf topos \( \mathcal{G} = \text{Set}^G \). In Sect. 4.1 below
we will consider this in more detail, but for the current Overview section, graphs can be understood as Graph-graphs. A graph \( G \) is finite if the set \( \text{Elem}(G) \) is finite. All finite graphs form a (sub)category (of \( G \)) denoted by \( G_{\omega} \).

**Predicate signatures and their models.**

**Definition 1 (Discrete Signatures and Mono Sketches).** Let \( G \) be a schema for graphs and \( G = \text{Set}^\mathcal{G} \) is the corresponding presheaf topos. A discrete predicate signature \( \mathcal{P} \) over \( G \) is given by a set \( |\mathcal{P}| \) of predicate symbols and an arity function \( \alpha: |\mathcal{P}| \rightarrow \text{Ob}(G) \). For \( P \in |\mathcal{P}| \), we say that graph \( \alpha(P) \) is the arity of \( P \), and we will often denote it by \( P^\alpha \). We will often (inaccurately) write \( \mathcal{P} \) for set \( |\mathcal{P}| \) too.

A (mono) sketch \( S \) over signature \( \mathcal{P} \) is given by a graph \( |S| \in G \) and a mapping \( \llbracket \cdot \rrbracket |S| \) such that diagram (2)(a) commutes.

\[
\begin{array}{ccc}
\mathcal{P} & \xrightarrow{\llbracket \cdot \rrbracket |S|} & 2 \text{Ob}(G) \\
\alpha & & \text{dom} \\
\text{Ob}(G) & \downarrow & \\
\end{array}
\quad
\begin{array}{ccc}
\mathcal{P} & \xrightarrow{\llbracket \cdot \rrbracket \mathcal{U}} & \text{Pred}(\mathcal{U}) \\
\alpha & & \text{dom} \\
\text{Ob}(G) & \downarrow & \\
\end{array}
\quad
(2)
\]

Given two sketches \( S, S' \), a morphism \( f: S \rightarrow S' \) between them is a graph morphism \( |f|: |S| \rightarrow |S'| \) such that for any \( d \in |P|_S \), we have \( d \circ f \in |P|_{S'} \), i.e., postcomposition with \( f \) maps diagrams in \( S \) to diagrams in \( S' \) with the same label.

**Corollary 1 (Sketch category).** It is easy to see that composition \( |f| \circ |f'|: |S| \rightarrow |S'| \rightarrow |S''| \) is a sketch morphism as soon as \( f \) and \( f' \) are such, and an identity graph morphism is a sketch morphism. This gives us a category \( G|\mathcal{P} \) of sketches over \( \mathcal{P} \), which is supplied with an obvious forgetful functor \( |\cdot|: G|\mathcal{P} \rightarrow G \). This functor is faithful and surjective on objects (any graph is a sketch with all diagram sets empty) but not full.

**Syntax and Semantics via Sketches.** For sketches of syntactical origin, graph \( X = |S| \) can be understood as a graph of variables typically countable or finite, and a diagram \( d \in |P| \), i.e., a graph morphism \( d: P^\alpha \rightarrow X \), can be seen as substituting variables from \( X \) for placeholders in \( P^\alpha \). In MDE, pairs \((P,d)\) are called constraint declarations (e.g., see Example in Fig. 16). For an ordinary string-based logic where graphs are (multi-sorted) sets, a diagram is a tuple of variables \((x_1, \ldots, x_i, \ldots)\), \( x_i = d(e_i) \), for some enumeration \( \{e_1, \ldots, e_i, \ldots\} \) of the arity \( P^\alpha \), and a labelled diagram \((P,d)\) is nothing but a formula \( P(x_1, \ldots, x_i, \ldots) \).

Two types of sketch semantics should be distinguished: elementwise (or value-oriented in terms of relational databases) and setwise (object-oriented as is typical for MDE). For the former, elements of the carrier \(|U|\) of a sketch \( U \) of semantic origin are values and links between them, and labelled diagrams are thus labelled tuples \( P(u_1, \ldots, u_i, \ldots) \) representing true facts about the model \( U \). Sets of such tuples are predicates, and the diagram (2)(a) above is re-denoted as shown in diagram (b), where \( \text{Pred}(U) \) is the set of all \( P_X \subseteq \text{G}(X,U) \) such that \( X \in \text{Ob}(G) \) and \( \text{dom}(P_X) = U \).
3. Diagram Logic and Algebra: Overview

For the setwise semantics, the carriers are graphs rather than sets, there are just few sketches of semantics origin such as $\textbf{Set}$ or $\textbf{Rel}$ or $\text{Span}(\textbf{Set})$ considered as sketches, whereas individual user-defined models are sketch morphisms into these. The following example clarifies these remarks.

**Example 1 (Elementwise vs. Setwise Logics).** Let $R \subseteq A \times B \times A$ be a two-sorted ternary predicate symbol, and let $R[\square] \subseteq \{A\} \times \{B\} \times \{A\}$ be a relation given elementwise, i.e., as a set of triples $(a_1, b, a_2) \in \{A\} \times \{B\} \times \{A\}$. This gives us a model of signature $\mathcal{P} = \{R\}$ further referred to as $\square$.

In our diagram logic, the graph schema is a two-element set (of sorts), $\mathcal{G} = \{A, B\}$, and a graph $G$ is thus a pair of sets $(G(A), G(B))$. Arity $R^\alpha$ is the following graph: $R^\alpha(A) = \{a_1, a_2\}$ and $R^\alpha(B) = \{b\}$, where $a_1, a_2, b$ are some chosen “placeholders”. Although $R^\alpha$ is a pair of sets, it is convenient to specify it as a labelled triple $(a_1, A; b, B; a_2, A)$.

A $\mathcal{R}$-model $\square$ described above is almost immediately a sketch $S[\square]$ whose carrier graph $U := |S[\square]|$ is a pair of sets, $U(A) = \{A\}$, $U(B) = \{B\}$, and the diagram set is defined as follows:

$$\left[ R \right]_{S[\square]} = \left\{ d \in G(R(U), U) \mid (d(a_1), d(b), d(a_2)) \in R[\square] \right\}$$

so that diagrams in $S[\square]$ are triples in $R[\square]$ (i.e., true facts about model $\square$).

Consider now a constraint for the model $\square$, say, a universal first-order sentence $\forall \phi$ that states $(\forall x \in A, \forall y \in B)R(x, y, x)$ (so that $\phi$ is formula $R(x, y, x)$). To specify this in our diagram logic, we introduce a formal syntactical sketch $S[\phi]$, with the carrier graph $X := |S[\phi]|$ defined by setting $X(A) = \{x\}$, $X(B) = \{y\}$ and the only (formal) diagram $d$ in $\left[ R \right]_{S[\phi]}$ is defined by formula $R(x, y, x)$ (i.e., $d(a_1) = d(a_2) = x$ and $d(b) = y$). It is easy to see that the standard definition of the satisfiability relation, $\square \models \forall \phi$, actually means that any graph morphism $f: X \to U$ (i.e., a mapping with $f(x) \in \{A\}$, $f(y) \in \{B\}$) is a correct sketch morphism that maps the only formal true fact in $S[\phi]$ to a true fact in $S[\square]$, that is, $(f(x), f(y), f(x)) \in R[\square]$. This is exactly the first-order logic semantics of universal formulas: any evaluation of variables results in a true fact about the model. Thus,

$$\square \models \forall \phi \quad \text{iff} \quad G[\text{ew}](S[\phi], S[\square]) \cong G[\text{ew}](|S[\phi]|, |S[\square]|)$$

(3)

where the superscript `ew` stands for ‘elementwise’: we added it to distinguish the above from the setwise framework considered below. Recall that $G[\text{ew}] = \{A, B\}$.

In the setwise sketch logic, relation $R$ is considered to be a sort rather than a predicate. It is a fundamental idea that changes everything. First, our carriers are now graphs with schema (1) and the carrier $X$ of syntactical sketch $S[\phi]$ is a ternary span with apex $R$ and three projection legs $p_i: R \to A$, $i = 1, 2$, $q: R \to B$; in addition, we require these legs to be jointly monic and hence place predicate symbol $[\text{monic}_3]$ with arity being graph $\text{Span}_3$ into the signature, and include the corresponding diagram $d: \text{Span}_3 \to X$ into $S[\phi]$.

The semantic story begins with a functor $\square^{\text{ew}}: X \to \textbf{Set}$, which is required to be a $[\text{monic}_3]$-sketch morphism (as $\textbf{Set}$ is an obvious $[\text{monic}_3]$-sketch too). To ensure that relation $\square^{\text{ew}}$ satisfies formula $\forall \phi$, we add to the signature a predicate symbol $P_{\forall \phi}$ of arity $\text{Span}_3$, and add to sketch $S[\phi]$ one more constraint declaration (labelled diagram) described by formula $P_{\forall \phi}(p_1, q, p_2)$. Finally, we convert $\textbf{Set}$ into a $P_{\forall \phi}$-sketch by defining

$$\left[ P_{\forall \phi} \right]_{\textbf{Set}} = \{ d \in \textbf{Set}^{\text{Span}} \mid d \models \forall \phi \}$$
Now we can state
\[ [\cdot] = \forall \varphi \iff [\cdot]^{\text{sw}}: S^{\text{sw}} \rightarrow \mathbf{Set}_P \text{ is in } G^{\text{sw}} \mid P^{\text{sw}} \] (4)

where \( G^{\text{sw}} \) is \( \text{Graph} \) (see (1)), \( P^{\text{sw}} = \{[\text{monic}_3], P_\varphi\} \), and \( \mathbf{Set}_P \) is \( \mathbf{Set} \) considered as \( P \)-sketch. Note that the sketch treatment in (4) works for any constraint, not necessarily universal. Whatever logic is employed for writing a closed formula \( \psi \), we introduce a predicate \( P_\psi \) of the corresponding arity, and as satisfiability classifies models into valid and invalid, we can convert \( \mathbf{Set} \) (or other semantic universe) into a \( P \)-sketch by including into class \( [P_\psi]_{\mathbf{Set}} \) exactly those diagrams, which satisfy formula \( \psi \).

**Multi-diagram vs. unique-diagram sketches.** To be completed

If relation \([P]\) were stored in a database, duplication of its rows (triples as above) would normally be allowed and considered normal: typically, a row records a state of an object, and two different objects may have the same state at one database snapshot at some time moment and be different at another snapshot/time moment.

Duplication is useful in syntax too: See discussion in Sect. A.

### 3.2 Diagram Operations and Algebras: Basic Intuitions

Below we assume given some fixed category of sketches \( S = G \mid P \) (which, according to Makkai’s extended theorem, can be considered graphs).

**Definition 2 (Diagram Operation).** An (diagrammatic) operation is a pair \( (\phi, \alpha_\phi) \) with \( \phi \) the name (or the naming symbol) and \( \alpha_\phi: I_\phi \rightarrow E_\phi \) a sketch monic called \( \phi \)'s arity.\(^3\)

Let \( A \in S \) be a \( P \)-sketch and \( (\phi, \alpha_\phi) \) an operation. We say that \( A \) carries the operation \( \phi \), or \( A \) is an \( \phi \)-algebra, if for any sketch mapping \( b: I_\phi \rightarrow A \) (called binding) there is one and only one sketch mapping \([\phi]^{A}(b): E_\phi \rightarrow A \) (the result) such that the triangle \( (I_\phi, E_\phi, A) \) in diagram (5) below commutes:

\[ \alpha_\phi \circ [\phi]^{A}(b) = b. \] This **commutativity condition** is fundamental: it ensures that an operation does not change its input data.

\[ \text{(5)} \]

Given another algebra \( A' \) for \( \phi \), a sketch morphism \( f: A \rightarrow A' \) is called \( \phi \)-compatible iff \([\phi]^{A'}(b) \circ f = [\phi]^{A}(b \circ f)\) holds for every \( b: I_\phi \rightarrow A \).

\(^3\) The choice for letters \( I \) and \( E \) is motivated by considering \( I_\phi \) as specifying the Input sketch for operation \( \phi \), and \( E_\phi \) as specifying the scopE of \( \phi \); using \( S \) for scope would conflict with \( S \) for sketches. Another interpretation of \( E \) as referring to the rEsult of the operation.
Definition 3 (Discrete Operation Signatures and Algebras). A discrete operation signature \( \mathcal{O} \) over \( S = \mathbb{G}[P] \) is given by a set \( |\mathcal{O}| \) of operation symbols and an arity function \( \alpha: |\mathcal{O}| \to \text{Ob}([\to, S]) \). For a symbol \( \phi \in |\mathcal{O}| \), we say that monic \( \alpha(\phi): I_\phi \to E_\phi \) is the arity of \( \phi \). We will also denote this monic by \( \phi^* \) or \( IE_\phi \) and simply write \( \mathcal{O} \) for \( |\mathcal{O}| \).

A signature \( \mathcal{O} \) is finitary if \( P \) is finitary and sketches \( I_\phi, E_\phi \) are finite for all \( \phi \in \mathcal{O} \). This is what we have in applications, but many of our results do not depend on the signature’s (in)finitarity.

An algebra \( A \) for \( \mathcal{O} \) is given by a carrier sketch \( |A| \) and an operation
\[
[\phi]^A: S(I_\phi, |A|) \to S(E_\phi, |A|)
\]
for each \( \phi \in \mathcal{O} \) (which means commutativity of the corresponding triangle for each \( \phi \)). An algebra homomorphism is a sketch morphism compatible with all operations in \( \mathcal{O} \).

The following is obvious.

**Proposition 1 (Category of \( \mathcal{O} \)-algebras).** There is a category \( \mathbb{S}^{\mathcal{O}} \) whose objects are algebras with \( S \)-carriers and arrows are \( \mathcal{O} \)-homomorphisms.

**Example 2 (Precategories).** A precategory is a graph with an arrow composition and identity loops. We formalize this in our framework with the following signature: \( G = \text{Graph}, P = \varnothing, \) and \( \mathcal{O}_{\text{preCat}} = \{([\text{id}],[;]) \} \) with arities specified in Fig. 4. Then any \( \mathcal{O} \)-algebra is a precategory, and \( \mathcal{O} \)-homomorphisms are functors

**Example 3 (Universal algebra).** Consider \( G = 1 \) so that \( G \cong \text{Set} \), and \( P = \varnothing \) so that \( G||P \cong G \cong \text{Set} \). Every “classical” operation signature \( \Omega \) with an arity function \( \ar : |\Omega| \to \text{Nat} \) can be interpreted as a diagrammatic signature \( \mathcal{O}_\Omega = (|\Omega|, \alpha) \) such that for each symbol \( \omega \), its arity is given by the input set \( I_\omega = \ar(\omega) \) considered as a set, the scope set \( E_\omega = I_\omega + \{\ast\} \) and the obvious monic \( m_\omega: I_\omega \to E_\omega \). Any \( \mathcal{O}_\Omega \)-algebra \( A \) is an \( \Omega \)-algebra \( \hat{A} \): let \( n = \ar(\omega) \) and
Table of Contents

\( a \in A^\alpha \), then setting \( \omega \hat{A}(a) \overset{\mathrm{def}}{=} \omega \hat{A}(a)(\ast) \) defines an \( \Omega \)-algebra structure on \( |A| \). The converse construction from a classical \( \Omega \) algebra \( \hat{A} \) to a diagrammatic \( \Omega \) algebra \( \hat{A} \) is equally obvious: for any \( a \in A^\alpha \) and any \( x \in I_\omega \subset E_\omega \), \( \omega \hat{A}(x) \overset{\mathrm{def}}{=} a(x) \), and \( \omega \hat{A}(\ast) \overset{\mathrm{def}}{=} \omega \hat{A}(a) \). Moreover, \( \hat{A} \cong A \) and \( \hat{A} \cong A \). It is also easy to check that the categories \( \mathcal{S} \mathcal{O} \Omega \) and \( \Omega \mathcal{A} \mathcal{G} \) are isomorphic. Thus, the ordinary universal algebra is a specialization of the discrete diagrammatic algebra.

3.3 Reasoning via diagrammatic operations

A principal feature of diagram chasing is simultaneous derivation of both new elements and their properties. A typical example is “monics are stable under pull-backs”. Moreover, operations only deriving properties also make sense and can be seen as implications. For example, the passage from precategories to categories can be described by requiring precategories to carry three more diagrammatic operations ensuring associativity of the composition and left and right unitality of the identity. To see this, we first note that any diagram operation \( \phi \) with arity \( \phi^\alpha : I_\phi \rightarrow E_\phi \) gives rise to a diagram predicate \( \omega \phi \) with arity \( r \phi^s \alpha = E_\phi \). Thus, the operational signature \( O_{\mathcal{P}r\mathcal{E}C} \) gives us a predicate signature \( O_{\mathcal{P}r\mathcal{E}C}^E \).

\[
E_{\mathcal{A}ssoc} :=
\begin{tikzpicture}
  \node (a) at (0,0) [circle, fill, inner sep=2pt] {}; \node (b) at (1,0) [circle, fill, inner sep=2pt] {}; \node (c) at (0,-1) [circle, fill, inner sep=2pt] {}; \node (d) at (1,-1) [circle, fill, inner sep=2pt] {};
  \draw (a) -- (b) node [midway, left] \( a \rightarrow b \); \draw (c) -- (d) node [midway, left] \( c \rightarrow d \);
  \draw (a) -- (c) node [midway, left] \( 1[:1] \); \draw (b) -- (d) node [midway, left] \( 2[:2] \); \draw (c) -- (d) node [midway, left] \( p \); \draw (b) -- (c) node [midway, left] \( q \);
\end{tikzpicture}
\]

Fig. 5: Associativity of composition as a diagram operation \( \mathcal{A}ssoc \) with arity \( I_{\mathcal{A}ssoc} : I_{\mathcal{A}ssoc} \rightarrow E_{\mathcal{A}ssoc} \) with sketches \( I_{\mathcal{A}ssoc} \) and \( E_{\mathcal{A}ssoc} \) as specified in the figure (but elements \( p, 1, 2 \) are shown in the sketch \( E_{\mathcal{A}ssoc} \))

Now we can specify the associativity of composition by two operations. The first has arity shown in Fig. 5. In sketch \( I_{\mathcal{A}ssoc} \), the labelled diagram \( 1[:1] \) says that \( p = a; b \) and label \( 2[:2] \) says that \( r = p; c \). Sketch \( E_{\mathcal{A}ssoc} \) extends \( I_{\mathcal{A}ssoc} \) with three new elements: arrow \( q \) and two labelled diagrams \( 1[:1] \) and \( 2[:2] \), which say that \( r = a; (b; c) \). Hence, if we require a precategory to be also an \( \mathcal{A}ssoc \)-algebra, it would satisfy the implication \( (a; b); c \Rightarrow a; (b; c) \). In a similar way we can specify operation \( \mathcal{A}ssoc' \) imposing implication \( a; (b; c) \Rightarrow (a; b); c \). Hence, a precategory which is also an \( \{\mathcal{A}ssoc, \mathcal{A}ssoc'\} \)-algebra has its composition operation associative.

In a similar way, we can specify the left and right unitality laws for the identity operation. First, we need to extend our signature of predicate symbols.
4 Advanced Diagram Logic: Predicate Symbols with Dependencies

with symbols $\mathbb{[}=\mathbb{Node}]$ and $\mathbb{[}=\mathbb{Arr}]$ for equality of nodes and arrows: the arity of symbol $\mathbb{[}=\mathbb{Node}]$ is a pair of nodes and the arity of $\mathbb{[}=\mathbb{Arr}]$ is a pair of arrows with different sources and different targets; in addition, we have a dependency: equality of arrows implies equality of nodes. Obviously, any graph is automatically a sketch in this equality signature (this is a diagrammatic version of logic with equality). However, in our sketches below we will not use $\mathbb{[}=\mathbb{Node}]$ as equality of the corresponding nodes is automatic in the diagram.

\[ IE_{\text{leftUnit}} := \]

\[ \xymatrix{ a \ar@{->}[r]^{[id]} & x \ar[r]^{} & \bullet \ar[r]^{r} & y } \quad \xymatrix{ a \ar@{->}[r]^{[id]} & x \ar[r]^{[]} & \bullet \ar[r]^{r} & y } \]

Fig. 6: Left unitality of identity as a diagram operation leftUnit

The diagram in Fig. 6 specifies the required law. In sketch $I_{\text{leftUnit}}$, label $[\text{id}]$ declares arrow $a$ to be the identity of node $x$, and label $[\cdot]$ says that $r = a; b$. Sketch $E_{\text{leftUnit}}$ extends $I_{\text{leftUnit}}$ with one more labelled diagram that declares arrows $b$ and $r$ to be equal. In a similar way we can specify operation rightUnit. Hence, a precategory which is also a \{assoc, assoc', leftUnit, rightUnit\}-algebra is a category.

4 Advanced Diagram Logic: Predicate Symbols with Dependencies

4.1 Generalized Graphs

As our discussion in Sect. 2 suggests, we need a very general notion of graphs as presheaves $G: \mathcal{G} \to \textbf{Set}$ for a suitable schema category $\mathcal{G}$. In all our applications, graphs have hierarchical structure that allows their traversal from the top elements to the bottom. For example, ordinary directed multigraphs are given by schema $\mathcal{G} = \text{Graph}$ with $\text{Ob(}\text{Graph}) = \{\text{Node, Arrow}\}$ and two non-identity arrows $\text{src, trg}: \text{Arrow} \to \text{Node}$ so that object $\text{Arrow}$ is above $\text{Node}$. (Below we will show other examples below.) Hence, we require category $\mathcal{G}$ to be hierarchical in the following sense.

Hierarchical categories

Definition 4 (Hierarchical categories). We consider a natural number $N$ as a category

$[0, N) = \{0 \to 1 \to \ldots \to (N - 1)\}$

consisting of $N$ objects and $N - 1$ non-identity arrows.\(^4\)

\(^4\) The reason we start counting at 0 rather than 1 is to make the generalization for the first infinite ordinal $\omega$ straightforward: category $\mathcal{G}$ has all natural numbers $n < \omega$ as objects.
A hierarchical category (h-category in brief) of height \( N \) is given by a small category \( \mathbb{H} \) along with a surjective on objects functor \( \text{hei}: \mathbb{H} \to [0, N]^{\text{op}} \) so that all layers \( L_n(\mathbb{H}) = \text{hei}^{-1}(n) \) for \( n < N \) are not empty. We refer to \( N \) by \( \text{Hei}(\mathbb{H}) \). Note that if \( N = 0 \), then category \([0, N]\) is empty and we have an empty hierarchical category of height 0. Thus, \( \mathbb{H} = \emptyset \) iff \( \text{Hei}(\mathbb{H}) = 0 \).

Moreover, the following acyclicity condition holds for any two objects \( x, y \) in \( \mathbb{H} \): if \( \mathbb{H}(x, y) \neq \emptyset \), then either \( \text{hei}(x) > \text{hei}(y) \) or \( x = y \), but in the latter case we require \( \mathbb{H}(x, x) = \{\text{id}_x\} \). Thus, all \( \mathbb{H} \)-arrows go down from higher to lower layers, and the only arrows within a layer are identities. For an h-category \( \mathbb{H} \), we call elements of layer \( L_n(\mathbb{H}) \) n-cell types or n-(hyper)edge types and typically denote them by \( H \). We will also call 0-cell types node types.

**Corollary 2.** If \( \mathbb{H} \) is an h-category, then for any \( n < N \), the full subcategory \( \mathbb{H}_n \subset \mathbb{H} \) generated by objects of height less than \( n \) is an h-category as well. We have a cumulative chain of h-categories

\[
\mathbb{H}_0 \subset \mathbb{H}_1 \subset \ldots \subset \mathbb{H}_{N-1} = \mathbb{H}
\]

and for any object \( H \in \mathbb{H} \), \( \text{hei}(H) = \min \{n| H \in \mathbb{H}_n\} \).

**Definition 5 (Discrete h-categories).** A hierarchical category is called discrete if its only arrows are identities. However, a discrete h-category is more than a set: it is a set equipped with a height function, i.e., a layered set \( \mathbb{H} = \bigcup_{n< N} L_n(\mathbb{H}) \). We will also write \( L_{\leq n}(\mathbb{H}) \) for the set \( \bigcup_{0\leq i\leq n} L_i(\mathbb{H}) \).

**Definition 6 (Finitary hierarchical categories).** A h-category is called finitary if for any object \( x \) the set \( \mathbb{H}(x, \_\_ \_) \) is finite.

**Corollary 3 (Finite h-categories).** A finitary h-category \( \mathbb{H} \) is finite as soon as its object set \( \text{Ob}(\mathbb{H}) \) is finite.

**Remark 1 (Finitarity conditions).** Def. 6 is the first in the family of similar definitions, which require finiteness of some construct assigned to the notion at hand. If an h-category \( \mathbb{H} \) is finite, then it is finitary, but the converse is obviously not true, and we will often deal with infinite but finitary objects (signatures, sketches, graphs).

**Definition 7 (Graphs).** Let \( \mathcal{G} \) be an h-category. An \( \mathcal{G} \)-graph is a functor \( G: \mathcal{G} \to \text{Set} \), i.e., a presheaf over schema \( \mathcal{G} \). We will normally refer to such functors as merely graphs, but if we want to stress their distinction from ordinary directed multigraphs (which are functors \( G: \text{Graph} \to \text{Set} \) over schema specified in (1)), we will refer to them as \( \mathcal{G} \)-graphs or, sometimes, as ‘graphs’ in single quotes. For a graph \( G: \mathcal{G} \to \text{Set} \), elements of set \( G(H) \) with \( H \in \mathcal{G}_n \) are called n-cells (of type \( H \)), and if \( \mathcal{G}_n \) is not a singleton, we say that n-cells are multi-sorted. We write \( \text{Elem}(G) \) for the set \( \bigcup_{H \in \text{Ob}(\mathcal{G})} G(H) \).

The presheaf topos of all \( \mathcal{G} \)-graphs will be denoted by \( \mathcal{G} \), and we assume that throughout the paper the schema \( \mathcal{G} \) is arbitrary but fixed. A graph \( G \) is finite if \( \text{Elem}(G) \) is finite. All finite graphs form a (sub)category (of \( \mathcal{G} \)) denoted by \( \mathcal{G}_\omega \).

We will try to depict ‘graphs’ as graphical objects. For example, if object \( H \in \mathcal{G} \) has exactly two non-looping outgoing arrows, say, \( x, y \), we will call them source and target, and depict each element \( a \in G(H) \) as an arrow from element \( a.G(x) \) to element \( a.G(y) \).
Example 4 (Fancy Graph). Assume $\mathcal{G}$ is the category in Fig. 7(a) (identities omitted); and Fig. 7(b) depicts a graph given by the following functor $G : \mathcal{G} \to \text{Set}$:

$$
\begin{align*}
G(\text{Node}) &= \{P, Q, A, B\} & G(2-\text{src}) &= \alpha \mapsto f \\
G(\text{Edge}) &= \{f, g, i\} & G(2-\text{trg}) &= \alpha \mapsto g \\
G(2-\text{Edge}) &= \{\alpha\} & G(\text{src}^1) &= n \mapsto 0 \\
G(\text{Edge}_{(0,4)}) &= \{n\} & G(\text{trg}^2) &= n \mapsto 1 \\
G(\text{src}) &= \{f \mapsto A, g \mapsto A, i \mapsto Q\} & G(\text{trg}^3) &= n \mapsto A \\
G(\text{trg}) &= \{f \mapsto B, g \mapsto B, i \mapsto Q\} & G(\text{trg}^4) &= n \mapsto B
\end{align*}
$$

Elements of $G(0)$ can be called nodes, elements of $G(1)$ and $G(2)$ can be visualized as arrows (since the respective object in $\mathcal{G}$ has exactly two outgoing non-identity morphisms. There is an edge between edges, $\alpha$, and a hyperegde $n$ connecting four elements (“sources” of the hypereledge) as shown (as the respective object in $\mathcal{G}$ has more than two outgoing non-identity morphisms). Lines connecting a hypereedge to its sources will be called tentacles.

Fig. 7: Example of a higher-order graph
4.2 Signatures and Sketches

Classic Ehresmann’s sketches consider signatures that only contain classes of limit and colimit predicates whose semantics is given by the respective universal properties. Generalized sketches consider arbitrary predicates, whose semantics is defined externally. This allows for more compact specifications because one does not need to add auxiliary structures as it is often required with classical sketches, see, e.g., [20] and discussion in [13].

Definition 8 (Predicate Signature). Let $\mathcal{G}$ be a schema category for graphs and $\mathcal{G} = \mathbf{Set}^{op}$ is the corresponding presheaf topos. A predicate signature $\mathcal{P}$ over $\mathcal{G}$ is given by a h-category $|\mathcal{P}|$, whose objects are called predicate symbols and arrows are dependencies, and an arity functor $\alpha: |\mathcal{P}| \to \mathcal{G}^{op}$. Thus, if a symbol $P$ depends on symbol $P'$ via dependency $p: P \to P'$, then there is a graph morphism $p^\alpha: P^\alpha \to P'^\alpha$, where we write $X^\alpha$ for $\alpha(X)$ for an $X \in \text{Ob}(|\mathcal{P}|)$ but we will use the bracketed notation too. For a predicate symbol $P \in |\mathcal{P}|$, we say that graph $P^\alpha$ is the arity of $P$. We will often (inaccurately) use symbol $\mathcal{P}$ for both a signature and its carrier category $|\mathcal{P}|$.

A symbol $P$ is called finitary if its arity graph $P^\alpha$ is a finite graph, and a signature is finitary if all its symbols are such so that $\mathcal{G}^{op}$ can be considered as the codomain of functor $\alpha$. Below we assume finitarity as a default assumption.

Example 5 (Pullbacks over monics). Consider 'graphs' being ordinary graphs, and a predicate signature with three symbols, whose arities are as expected and shown in Fig. 8(b,c,d) resp. We assume these arities to be concrete fixed graphs whose elements are given fixed names formed
by numeric strings (but we could also use colours or whatever names we like). We assume that predicate \([pb_m]\) can only be declared for squares in which arrows 10 and 32 are declared to be \([\text{monic}]\). Later we will describe \([pb_m]\) as an operation for which the requirement for arrow 10 to be monic will be a precondition while a similar requirement for arrow 32 will be a postcondition, but when \([pb_m]\) is a predicate, both requirements are “postconditions”. Another postcondition is the requirement for the entire square to be \([\text{sqr}]\) (read “commutative”). Then we partition \(P\) into two layers, \(|P|_0\) “tr monic”, \(|P|_1\) “tr pb monic su” connected by dependency arrows \(p_1, p_2, p\) as shown in diagram (a). This gives us an h-category \(P\) and a functor \(\alpha: P \to G\) with \(p_\alpha(10) = 10, p_\alpha(32) = 32,\) and \(p_\alpha = \text{id}\) (or isomorphism if these squares are different).

We need predicate symbols to declare atomic constraints, e.g., \(P(x, y)\) in ordinary first-order logic, and simplest logical theories are built from conjunctions of atomic formulas, e.g., \(P(x, y) \land P(y, z)\). A diagrammatic counterpart of conjunctive theories is the notion of a sketch: a set of atomic formulas over a graph of variables, which is closed w.r.t. all dependency arrows in the signature.

Definition 9 (Sketch over a signature \(P\) with dependencies). A (generalized) sketch \(S\) over \(P\) is given by a graph \(|S| \in G\) called the carrier, and a span of two functors \(\text{label}\) and \(\text{diagr}\) making a commutative outer square in diagram (7). The apex of the span is a small category \(\text{Index}(S)\) of (diagram) indexes and dependencies; moreover, functor \(\text{label}: \text{Index}(S) \to |P|\) is required to be a discrete opfibration so that all arrows in \(\text{Index}(S)\) are lifts of \(P\)-arrows—this is the closure condition mentioned above (it is discussed in detail in [13]).

We call the definition above fibrational to distinguish it from a typical indexed sketch definition, in which the discrete opfibration \(\text{label}\) is replaced by presheaf \([\_]\): \(|P| \to \text{Set}\) and \(\text{diagr}\) is a functor \([\_]: |S| \to G^{op}/S\) (this is how sketches were defined in [13] assuming also that \(\text{diagr}\) is injective).

Corollary 4 (Sketches are fibration morphisms). As \(\text{dom}_{|S|}^{op}\) is a discrete fibration (see Lemma 1), commutativity of the outer square in diagram (7) makes the pair \((\alpha, \text{diagr})\) an opfibration morphism. This is an exact formulation of the major requirements for sketches with dependencies: the set of diagrams is to be closed under “inference rules” arising from dependencies.

Definition 10 ((Multi)sketches vs. mono-sketches). Let the internal square in diagram (7) be a pullback in \(\text{Cat}\), then its apex is the category whose objects are all \(P\)-labelled diagrams over graph \(|S|\) and arrows are lifts of dependency arrows in \(P\) (recall that pulling back a split fibration results in a split fibration). Objects of \(\text{Index}(S)\) can be identified with indexed pairs \((P, d)\) with \(P\) a predicate symbol and \(d: P^\alpha \to |S|\) a diagram. In the MDE parlance, pairs \((P, d)\)
A sketch is called mono-sketch if the pair of functors \((\text{label}, \text{diagr})\) is jointly monic and hence mapping \# is monic.

**Definition 11 (Finite Sketches).** A sketch \(S\) is carrier-wise finite if the graph \(|S|\) is finite, and constraint-wise finite if the set \(\text{Index}(S)\) is finite. A sketch is finite if both finiteness conditions hold.

**Definition 12 (Sketch morphisms).** Let \(S = (|S|, \text{label}, \text{diagr})\) and \(S' = (|S'|, \text{label}', \text{diagr}')\) be two sketches. A sketch morphism \(f: S \to S'\) is a pair consisting of an \(\mathcal{G}\)-arrow \([f]: |S| \to |S'|\) and a functor \(f_{\text{const}}: \text{Index}(S) \to \text{Index}(S')\) such that \(f_{\text{const}} \circ \text{label}' = \text{label}\) and \(f_{\text{ind}} \circ \text{diagr}' = \text{diagr} \circ \text{label}\). This gives us the category of \(\mathcal{P}\)-sketches denoted by \(\mathcal{G}|\mathcal{P}\) (this is Makkai’s notation that turned out convenient).

\[
\begin{array}{c}
\text{Index}(S') \\ \downarrow \text{diagr}' \\
\mathcal{G}^{op}/|S'| \\
\downarrow \text{label}'
\end{array}
\xrightarrow{\text{f}_{\text{const}}} 
\begin{array}{c}
\text{Index}(S) \\
\downarrow \text{diagr}
\end{array}
\xleftarrow{\text{label}}
\begin{array}{c}
|\mathcal{P}| \\
\downarrow \alpha
\end{array}
\xrightarrow{\text{dom}^{op}}
\begin{array}{c}
\mathcal{G}^{op}/|S| \\
\downarrow \text{dom}^{op}
\end{array}
\xrightarrow{\text{f}_{\text{ind}}}(8)
\]

Note that if the target sketch is a mono sketch, then the second component of a sketch morphism becomes redundant as it can be derived from \([f]: |S| \to |S'|\); given \(f \in \text{Index}(S)\) with \(P = \text{label}(I)\) and \(d = \text{diagr}(I): P \to |S|\), we define \(f_{\text{ind}}(I)\) to be the index over \(|S'|\) determined by the pair \((P, d \circ f)\).

### 4.3 Makkai’s Construction: Sketches as Graphs

A fundamental and not obvious property of the sketch construct (first noticed by Makkai for the case of a discrete signature without dependencies [25]) is that multisketches are, in fact, presheaves — Makkai justified the notion of a multisketch by exactly this reason. However, as we have seen above, the notion of a multisketch has quite practical reasons to be considered: mathematical models often turn out smarter than their creators could foresee.

In this section we will extend Makkai’s theorem for the case of predicate signatures with dependencies, whose proof, unexpectedly, turned out to be much more complex. We begin with a simple example to illustrate the idea and then give a general proof.

**Example (Example 5 cont’d )** Assume signature \(\mathcal{P}_{\text{pb}} = \{\text{monic}, [\text{pb}], [\text{=sr}]\}\) described in Example 5 on p. 20, which is defined over h-graphs with schema \textbf{Graph} described in diagram (1) on p. 11. We want to build a metamodel (metasketch, schema) for \(\mathcal{P}_{\text{pb}}\)-sketches, that is, a category \(S(\mathcal{P}_{\text{pb}})\) such that any \(\mathcal{P}_{\text{pb}}\)-sketch could be seen as a functor \(S: S(\mathcal{P}_{\text{pb}}) \to \text{Set}\).
with binding maps, e.g., we require that dependant diagrams are obtained by postcomposition of inter-arity maps (Fig. 8), and we require commutativity of the corresponding triangles to ensure (see Fig. 8(c)).

Indeed, the coproduct whose names are over-lined names of the four arrows in the arity graph. Moreover, we require \( S \) in \( \mathcal{P} \) to be included into \( \mathcal{P}_{pbm} \), and two vertical arrows between them in Fig. 9) so that category \( \mathcal{P}_{pbm} \) and \( \mathcal{P} \) are, resp., blue and orange, binding (wavy) arrows are black.

The carrier of sketches are specified by schema \( \text{Graph} \) (the top two nodes and two vertical arrows between them in Fig. 9) so that category \( \text{Graph} \) is to be included into \( \mathcal{P}_{pbm} \). We enrich this schema with three new nodes \([\text{monic}], [\text{pb}], [=\text{sqr}]\) as shown in Fig. 9, which in a \( \mathcal{P}_{pbm}\)-sketch \( S \) will be instantiated by the corresponding constraints. To ensure that elements of sets \( S(x), x \in \{[\text{monic}],[\text{pb}],[=\text{sqr}]\} \) are indeed diagrams of the corresponding shapes, we add to constraint nodes suitable \( \text{binding} \) arrows, e.g., arrows \( 01, 1; [\text{monic}] \rightarrow \text{Node} \) ensure that any constraint labelled \([\text{monic}]\) will refer to the two nodes in the carrier graph \([S]\) corresponding to the two nodes in the arity graph \([\text{monic}]\) (see Fig. 8(b)), and arrow \( 01; [\text{monic}] \rightarrow \text{Edge} \) ensures that any \([\text{monic}]-\)constraint in \( S \) will refer to an arrow in the carrier graph corresponding to arrow \( 01 \) in the arity graph. Moreover, we require \( 01; \text{src} = 0 \) and \( 01; \text{trg} = 1 \) so that any functor \( S \) necessarily respecting these constraint would make any element in set \( S([\text{monic}]) \) a correct diagram \( b_s; [\text{monic}] \rightarrow |S| \).

We repeat this trick for two other constraint nodes \([\text{pb}], [=\text{sqr}]\) as shown in the figure. To simplify drawing, we use the following visualization: if an arrow is labelled by a set, it actually abbreviates a set of parallel arrows — one arrow per label, e.g., the vertical arrow from \([\text{pb}]\) to \( \text{Edge} \) abbreviates four arrows whose names are over-lined names of the four arrows in the arity graph \([\text{pb}]\) (see Fig. 8(c)).

Arrows \( p_1,p_2,p \) are the three dependency arrows in the signature \( \mathcal{P}_{pbm} \) (see Fig. 8), and we require commutativity of the corresponding triangles to ensure that dependant diagrams are obtained by postcomposition of inter-arity maps with with binding maps, e.g., we require \( p_1;01 = 00 \) and \( p_2;01 = 02 \). This gives us a category \( \mathcal{P}_{pbm} \) consisting of three parts: a) category \( \text{Graph} \) is embedded into \( \mathcal{P}_{pbm} \), b) category \( \mathcal{P}_{pbm} \) is embedded, and c) a set of intermediate binding arrows from \( \mathcal{P}_{pbm}\)-objects to \( \text{Graph}\)-objects such that some commutativity conditions hold.

It is easy to see that any functor \( S; \mathcal{P}_{pbm} \rightarrow \textbf{Set} \) is actually a \( \mathcal{P}_{pbm}\)-sketch \( \hat{S} \). Indeed, the coproduct \( \bigsqcup_{x \in \text{Ob}(\mathcal{P}_{pbm})} \hat{S}(x) \) gives us the sketch’s set of constraints \( \text{Index}(\hat{S}) \) with an obvious mapping \( \text{label}: \text{Index}(\hat{S}) \rightarrow \text{Ob}(\mathcal{P}_{pbm}) \). Moreover, functions \( \hat{S}(x), x \in \{p_1,p_2,p\} \) give us discrete opcartesian lifts of the three dependen-
cies in $P_{\text{pb}_a}$ so that label becomes a discrete opfibration. Finally, binding arrows attached to a node $X \in \{\text{monic}, [\text{pb}], [=_{op}]\} \subset P_{\text{pb}_a}$ along with commutativity conditions ensures a correct binding mapping for every constraint $x \in \hat{S}(X)$, and due to the universal property of coproducts, we have a correct binding functor $\text{diagr}: \text{Index}((\hat{S})) \to \hat{G}/[\hat{S}]$. It is also easy to see that the converse is true as well: any $P_{\text{pb}_a}$-sketch $S$ gives rise to a functor $\hat{S}: P_{\text{pb}_a} \to \text{Set}$ such that the two transformations are mutually inverse (below we will prove it for the general case).

### Signatures as profunctors and their collages

Given a schema $\mathcal{G}$ for graphs, a predicate signature over $\mathcal{G}$ is a functor $\alpha: \mathcal{P} \to (\text{Set}^{\mathcal{G}})^{\text{op}}$, whose opposite is $\alpha^{\text{op}}: \mathcal{P}^{\text{op}} \to \text{Set}^{\mathcal{G}}$. Currying gives us a profunctor

$$\bar{\alpha}: \mathcal{G} \times \mathcal{P}^{\text{op}} \to \text{Set}.$$  \hspace{1cm} (9)

which maps a pair of objects $H \in \mathcal{G}$, $P \in \mathcal{P}$ to the set

$$\bar{\alpha}(H, P) = \{\pi | a \in P^{\alpha}(H)\}$$  \hspace{1cm} (10)

where writing $\pi$ is a notational trick to show the change of the role of element $a$ from being an element of an arity shape to being a binding arrow $\pi: H \leftarrow P^{\beta}$; note that arrows go from the second component of the pair to the first one). In the opposite direction, we write $\underline{a} \in P^{\alpha}(H)$ for a given binding arrow $a \in \bar{\alpha}(H, P)$, thus,

$$P^{\alpha}(H) = \{\underline{a} | a \in \bar{\alpha}(H, P)\}$$  \hspace{1cm} (11)

and $\underline{a} = a$ for any $a \in \bar{\alpha}(H, P) \sqcup P^{\alpha}(H)$.

It is known that any profunctor is equivalent to a cospan of categories

$$\mathcal{G} \overset{\bar{\alpha}}{\underset{\mathcal{P}}{\leftrightarrow}} \mathcal{G}^{\star} \cdot \mathcal{P}$$  \hspace{1cm} (12)

whose apex is the collage category of the profunctor and legs are full embeddings [?][Benabou].

Add citation: 1) Distributors at Work by Jean Bénabou June 2000

2) Joyal’s CatLab Distributors and barrels. The collage category is defined as follows. $\text{Ob}(\mathcal{G}^{\star} \cdot \mathcal{P}) = \text{Ob}(\bar{\mathcal{G}}) \sqcup \text{Ob}(\mathcal{P})$, and for a pair of objects $X, Y \in \text{Ob}(\mathcal{G}^{\star} \cdot \mathcal{P})$, we set

$$\mathcal{G}^{\star} \cdot \mathcal{P}(Y, X) = \begin{cases} \mathcal{P}(X, Y) & \text{if } X, Y \in \mathcal{P} \\ \mathcal{G}(X, Y) & \text{if } X \in \mathcal{G} \\ \bar{\mathcal{G}}(Y, X) & \text{if } Y \in \mathcal{G} \text{ and } X \in \mathcal{P} \\ \emptyset & \text{if } Y \in \mathcal{P} \text{ and } X \in \mathcal{G} \end{cases}$$  \hspace{1cm} (13)

Note that for category $\mathcal{G}^{\star} \cdot \mathcal{P}$, the usual notation for hom-sets is reversed and formula $\mathcal{G}^{\star} \cdot \mathcal{P}(\_, P)$ denotes the set of arrows from $P$. Arrow composition is given by functions $\bar{\alpha}(h, id)$ and $\bar{\alpha}(id, p)$ as shown in the inset figure, and its associativity is ensured by the functoriality of $\alpha$. Arrows in $\bar{\alpha}(X, Y)$ are sometimes called formal or wavy; in our context, they are exactly binding arrows we considered in the example above.

It is easy to see that the collage category above is an instance of the following construct.

5 with a more formal attitude, we can consider $\pi$ as a pair $(a, *)$ with * being some predefined token.
Definition 13. A split h-category is a pair \((\mathbb{H}, m)\) with \(\mathbb{H}\) a hierarchical category of height \(N \geq 2\) and \(m \in (0, N)\) a natural number. Objects of height less than \(m\) (there is at least one) are called cell types or graphical; they and arrows between them form a full category \(\mathbb{H}_{[0,m]} \subset \mathbb{H}\). Objects of height more or equal to \(m\) (there is at least one) are predicate symbols (logical elements); they and arrows between them form a full subcategory \(\mathbb{H}_{[m,N]} \subset \mathbb{H}\). Arrows from predicate symbols to cell types are called *wavy*.

Proposition 2. The constructs of a predicate signature and a split h-category are naturally equivalent.

Proof. Let \((\mathbb{H}, \text{hei}, m)\) is a split h-category of height \(N\). The intermediate \(m\) generates a functor \(m: [0, N - 1] \rightarrow [0, 1]\) whose composition with \(\text{hei}\) gives us a barrel \(b = \text{hei} \circ m: \mathbb{H}_{\text{op}} \rightarrow [0, 1]\) and hence a profunctor \(\hat{b}: \mathbb{H}_{[0,m]} \times \mathbb{H}_{\text{op}}^{[m,N]} \rightarrow \text{Set}\) (see Joyal [8]); uncurrying the latter gives us a signature \(\alpha_m: \mathbb{H}_{\text{op}}^{[m,N]} \rightarrow \text{Set}^{\mathbb{H}_{[0,m]}}\). The opposite transformation was considered above. Natural isomorphism of exponentiation and currying, and of profunctors and barrels, give us required natural equivalence. Thus, we can consider any split h-category as the collage \(\mathcal{G} \star_{\alpha} \mathcal{P}\) of a graphical schema \(\mathcal{G} = \mathbb{H}_{[0,m]}\) and a signature \(\mathcal{P} = \mathbb{H}_{[m,N]}\) with arity \(\alpha_m\) defined as above.

This results is the first step in our general program of showing that distinction between graphs and theories over them is conventional. The second (and final) step is given by the following result.

Theorem 1 (Extended Makkai’s Theorem). The category of \(\mathcal{P}\)-sketches \(\mathcal{G} \mid \mathcal{P}\) is equivalent to the presheaf topos \(\text{Set}^{\mathcal{G} \star_{\alpha} \mathcal{P}}\). In other words, \(\mathcal{P}\)-sketches over \(\mathcal{G}\)-graphs can be interpreted as \((\mathcal{G} \star_{\alpha} \mathcal{P})\)-graphs (and thus diagrammatic constraints are just higher-order cells).

Proof (Sketch) We will show equivalence by building two mutually inverse functors.

From presheaves to sketches. Having a presheaf (graph) \(G \in \text{Set}^{\mathcal{G} \star_{\alpha} \mathcal{P}}\), we build a sketch \(\mathcal{S}(G)\) with

(a) \(|\mathcal{S}(G)| = \text{proj}_{\mathcal{G}} = i_{\mathcal{G}} \downarrow G\), and obviously we have a naturally bijective correspondence between these parts of presheaf on \(\mathcal{G} \star_{\alpha} \mathcal{P}\) and \(\mathcal{G} \mid \mathcal{P}\)-sketch structures.

(b) We have a presheaf (indexed set/family) \([ \_ ]_{\mathcal{S}(G)} = \text{proj}_{\mathcal{P}} = i_{\mathcal{P}} \downarrow G\), and define label\(_{\mathcal{S}(G)}\) to be the respective discrete opfibration (see, e.g., Emily Riehl’s survey).

(c) it remains to build the diagram functor \(\text{diag}_{\mathcal{S}(G)}: [\_ ]_{\mathcal{S}(G)} \rightarrow \mathcal{G} \mid |\mathcal{S}(G)|\) that makes a commutative square as needed, and show that the construction provides a natural isomorphism between presheaves and sketches. This is the most difficult part, but it’s managed by the Yoneda lemma. The latter states that for any predicate symbol \(P\), we have an isomorphism natural wrt. both \(P\) and \(G\):

\[
G(P) \cong_{P,G} \text{Set}^{\mathcal{G} \star_{\alpha} \mathcal{P}}(\mathcal{G} \star_{\alpha} \mathcal{P}(\_ P), G)
\] (14)

(recall that for category \(\mathcal{G} \star_{\alpha} \mathcal{P}\), the usual notation for hom-sets is reversed and formula \(\mathcal{G} \star_{\alpha} \mathcal{P}(X, P)\) denotes the set of arrows from \(P\) to \(X\)). We need to unpack this statement in our context for our notation.
We begin with the following observation. For any predicate \( P \), the hom-functor \( G \star \alpha P(\_\_ P) \) actually works as follows:

\[
G \star \alpha P(x, P) = \begin{cases} 
  P^\alpha(x) & \text{if } x \in \text{Elem}(G) \\
  \mathcal{P}(P, \_\_)(x) & \text{if } x \in \text{Elem}(\mathcal{P}) \\
  x: P^\alpha(H) \leftarrow \{\ast\} & \text{if } x: H \leftarrow P \text{ is a wavy arrow}
\end{cases}
\]

(recall that \( \mathcal{P}(P, P) = \{\text{id}_P\} \) for each \( P \) and thus any wavy arrow from \( P \) is an element in \( P^\alpha \)).

Applying (15) to Yoneda (14) for \( p: P \rightarrow P' \) and restricting each of the natural transformations in the set to objects of subcategory \( \mathcal{G} \), we get the following family of commutative diagrams indexed by \( p \):

\[
\begin{array}{ccc}
G(P) & \overset{\text{Yoneda}}{\cong} & \mathsf{Set}^{G \star \alpha P}(G \star \alpha P(\_\_ P), G) \\
& \downarrow{\cong} & \downarrow{\alpha}
\end{array}
\quad\quad
\begin{array}{ccc}
\mathsf{Set}^{\mathcal{G}}(P^\alpha, \mathcal{P}) & \overset{\text{diagr}_{\mathcal{G}}}{\rightarrow} & \mathsf{Set}^{\mathcal{G}}(P^\alpha, \mathcal{P})
\end{array}
\]

whose right square gives us the sketch-structure view of diagrams extracted from the presheaf \( G \).

Applying (15) to Yoneda (14) for \( q: G \rightarrow G' \) and again restricting each of the natural transformations in the set to objects of subcategory \( \mathcal{G} \), we get another family of commutative diagrams indexed by \( q \):

\[
\begin{array}{ccc}
G(P) & \overset{\text{Yoneda}}{\cong} & \mathsf{Set}^{G \star \alpha P}(G \star \alpha P(\_\_ P), G) \\
& \downarrow{\cong} & \downarrow{\alpha}
\end{array}
\quad\quad
\begin{array}{ccc}
\mathsf{Set}^{\mathcal{G}}(P^\alpha, \mathcal{P}) & \overset{\text{diagr}_{\mathcal{G}}'}{\rightarrow} & \mathsf{Set}^{\mathcal{G}}(P^\alpha, \mathcal{P})
\end{array}
\]

which shows that translation of presheaves to sketches is functorial.

From sketches to presheaves. Having a sketch \( S \in \mathcal{G}||\mathcal{P} \), we build a presheaf (graph) \( G^S \) with

(a) \( \text{proj}_{\mathcal{G}} = |S| \), and obviously \( |G^S| = |S| \).

(b) \( \text{proj}_{\mathcal{P}} = \prod_{\text{s.label}} \) is the presheaf corresponding to the discrete opfibration component of \( S \) — functor \( S.\text{label} \). As is well known, the correspondence between presheaves and discrete opfibrations is naturally bijective (it's just a special discrete case of the Grothendieck construction).

(c) it remains to define functions \( G^S(\bar{x}) \) for each wavy arrow \( \bar{x}: H \leftarrow P \), and we set for any constraint index \( c \in [P] \),

\[
G^S(c) = (S.\text{diagr})(c)(a).
\]

Recall that \( \text{diagr}(c) \) (here and below we will omit attribution to \( S \)) is a diagram of shape \( P^\alpha \), i.e., a natural transformation \( \text{diagr}(c): P^\alpha \Rightarrow |S| \) of functors into \( \mathsf{Set} \), i.e., a family of functions \( \text{diagr}(c)_H \) for each \( H \in \mathcal{G} \), and \( a \) is the element in \( P^\alpha(H) \) corresponding to the wavy arrow \( a \in a(\bar{x}(P, H)) \) (see (10)).
4 Advanced Diagram Logic: Predicate Symbols with Dependencies

For further references, we summarize the above as shown below (we write `label` for `S.label` and `diagr` for `S.diagr`):

\[
G^S(x) = \begin{cases} \{S(x)\} & \text{if } x \in \text{Elem}(G) \\ \langle x \rangle_{\text{label}} & \text{if } x \in \text{Elem}(P) \\ \{c \mapsto \text{diagr}(c)_{\text{H}(x)}: c \in \langle P \rangle \} & \text{if } x \in \overline{\alpha}(P, H) \end{cases} 
\]

(18)

We need to check the functoriality of so defined functions \(G^S(a)\) wrt. compositions of wavy arrows with predicate and cell arrows, i.e., for any triple of arrows \(p: P' \to P\), \(a \in \overline{\alpha}(P, H)\) and \(h: H \to H'\), compositions (i) \(a \circ h\) and (ii) \(p \circ a\) are preserved.

Case (i) is depicted in diagram (19)(a). As \(\text{diagr}(c)\) is a graph morphism for any predicate \(P\) and index \(c \in \langle P \rangle\), diagram (19)(b) also commutes for any \(H \in G\):

\[
\begin{array}{ccc}
P & \xrightarrow{a} & H \\
\downarrow & & \downarrow \quad \downarrow \quad \downarrow \\
P' & \xrightarrow{a' = \overline{\alpha}(P, h)(a)} & H'
\end{array} \quad \begin{array}{ccc}
P^\alpha(H) & \xrightarrow{\text{diagr}(c)_{\text{H}(\text{P})}} & |S|(H) \\
\downarrow & & \downarrow \quad \downarrow \quad \downarrow \\
P'^\alpha(h) & \xrightarrow{\text{diagr}(c)_{\text{H}'(\text{P})}} & |S|(H')
\end{array} 
\]

(19)

Now we compute (we will write \(\overline{\alpha}(P, h)\) for \(\overline{\alpha}(\text{id}_P, h)\)):

\[
G^S(a \circ h)(c) = G^S(a')(c) \quad \text{by Def. of composition in } G \ast \ast_P \\
= \text{diagr}(c)_{\text{H}'(a')} \quad \text{by Def. of } G^S (18), \text{ 3rd case} \\
= \text{diagr}(c)_{\text{H}'}(P^\alpha(h)(a)) \quad \text{by Def. of } a \\
= |S|(h) \left( \text{diagr}(c)_{\text{H}(a)} \right) \quad \text{by commutativity (19)} \\
= G^S(h) \left( \text{diagr}(c)_{\text{H}(a)} \right) \quad \text{by Def. of } G^S (18), \text{ 1st case} \\
= G^S(h) \left( G^S(a)(c) \right) \quad \text{by Def. of } G^S (18), \text{ 3rd case} \\
= \left( G^S(a) \circ G^S(h) \right)(c)
\]

Since the above holds for all \(c \in \langle P \rangle_{\text{label}}\), we conclude that \(G^S(a \circ h) = G^S(a) \circ G^S(h)\).

Case (ii) is in diagram (20)(a). For any \(c' \in \langle P' \rangle\), let \(\overline{c'}: c' \to c\) be the lifting of arrow \(p: P' \to P\) in \(P\) at \(c' \in \text{Index}(S)\), i.e., \(\langle p \rangle(c') = c\). As \(\text{diagr}(p)\) is a diagram morphism, i.e., a commutative triangle, then diagram (20)(b) also commutes for any \(H \in G\):

\[
\begin{array}{ccc}
P & \xrightarrow{a} & H \\
\downarrow & & \downarrow \quad \downarrow \quad \downarrow \\
P' & \xrightarrow{a' = \overline{\alpha}(P, h)(a)} & H
\end{array} \quad \begin{array}{ccc}
P^\alpha(H) & \xrightarrow{\text{diagr}(c)_{\text{H}(\text{P})}} & |S|(H) \\
\downarrow & & \downarrow \quad \downarrow \quad \downarrow \\
P'^\alpha(h) & \xrightarrow{\text{diagr}(c')_{\text{H}'(\text{P})}} & |S|(H')
\end{array} 
\]

(20)
Now we compute:

\[ G^S(p ; a)_c(c') = G^S(a')(c') \text{ by Def. of composition in } \mathcal{G}_a\mathcal{P} \]
\[ = \text{diagr}(c')_H(a'_1) \text{ by Def. of } G^S (18), 3rd case \]
\[ = \text{diagr}(c')_H(p^H_2(a)) \text{ by Def. of } \alpha \]
\[ = \text{diagr}(c')_H(a) \text{ by commutativity (20)} \]
\[ = G^S(a)(c) \text{ by Def. of } G^S (18), 3rd case \]
\[ = G^S(a) \left( G^S(p)(c') \right) \text{ by Def. of } G^S (18), 2nd case \]
\[ = \left( G^S(p) \; G^S(a) \right)(c') \]

Since the above holds for all \( c' \in \mathbb{P}_{\text{label}} \), we conclude that \( G^S(p ; a) = G^S(p) \; G^S(a) \).\(^6\)

Thus, we have built a presheaf \( G^S \in \text{Set}^{\mathcal{G}_a\mathcal{P}} \) from sketch \( S \in \mathbb{G}||\mathcal{P} \) and checking functoriality of sketch-to-presheaf translation is straightforward (commutativity of diagram mapping with a sketch morphism is exactly naturality of presheaf morphism wrt. wavy arrows). It is easy to see that the two translations are exactly mutually inverse: \( S(G^S) \cong S \) and \( G^S(G) \cong G \). Yoneda lemma now implies that categories \( \text{Set}^{\mathcal{G}_a\mathcal{P}} \) and \( \mathbb{G}||\mathcal{P} \) are naturally isomorphic.

### 4.4 Discrete Hierarchical Signatures and Sketches

We will redefine the notions of a predicate signature with dependencies and sketches over it in a discrete (but hierarchical) way: it is somewhat unexpected that this partial de-categorification does not actually lose information and the semi-discrete (below) and the fully categorical (above) notions are equivalent.

To distinguish the two families, we will refer to the definition of a signature with dependencies as \textit{functorial} (or \textit{f-signature}) and a corresponding notion of a sketch as \textit{fibtrational} (\textit{f-sketch}). Their counterparts defined below are \textit{dh-signature} and \textit{dh-sketch} with prefix ‘dh’ referring to \textit{discrete hierarchical}.

Also, in this section we will denote discrete signatures and categories of sketches over them (defined in Def. 8) by symbols \( \mathbb{P}^\bullet \) and \( \mathbb{S}||\mathcal{P}^\bullet \), where superscript \( \bullet \) is intended to recall that a signature carrier is a set rather than a category, and \( \mathbb{S} \) refers to the category of carriers which can be the category of graphs \( \mathbb{G} \), or some category of sketches defined before (recall that “sketches are graphs” by Makkai’s theorem). Finally, in this section we will begin counting layers of hierarchical constructs at 1 rather than 0 simply because it’s convenient to assign an empty signature of predicate symbols to index 0 rather than -1.

**Definition 14 (Dh-signatures and (dh-)sketches over them).** Let \( \mathbb{G} \) be a presheaf topos of \( \mathcal{G} \)-graphs as before. A \textit{predicate dh-signature of height} \( N \) is given by the inductively defined data specified by the commutative diagram

\[ \begin{array}{c}
G^S(p \; a) = G^S(a) \; G^S(p) \\
\end{array} \]

\[ \text{by Def. of composition in } \mathcal{G}_a\mathcal{P} \]

\[ = \text{diagr}(c')_H(a'_1) \text{ by Def. of } G^S (18), 3rd case \]

\[ = \text{diagr}(c')_H(p^H_2(a)) \text{ by Def. of } \alpha \]

\[ = \text{diagr}(c')_H(a) \text{ by commutativity (20)} \]

\[ = G^S(a)(c) \text{ by Def. of } G^S (18), 3rd case \]

\[ = G^S(a) \left( G^S(p)(c') \right) \text{ by Def. of } G^S (18), 2nd case \]

\[ = \left( G^S(p) \; G^S(a) \right)(c') \]

\[ \text{Since the above holds for all } c' \in \mathbb{P}_{\text{label}}, \text{ we conclude that } G^S(p ; a) = G^S(p) \; G^S(a). \]
Similarly, any wavy arrow (10) and (11) (Example in Fig. 9 shows how it works). We continue for (22):

\[
\emptyset \rightarrow \mathcal{P}^\bullet_0 \rightarrow \mathcal{P}^\bullet_1 \rightarrow \ldots \rightarrow \mathcal{P}^\bullet_{N-1}
\]

(21).

The upper row is provided by a discrete hierarchical category \(\mathcal{P}^\bullet\) of predicate symbols of height \(N\), i.e., a finite chain of non-empty sets shown in the top row of the diagram. Each of these sets \(\mathcal{P}^\bullet_i\) is equipped with an arity function \(\alpha^\bullet_i\) into the corresponding category of sketches \(S_{i-1} \overset{\text{def}}{=} S_{i-2}||\mathcal{P}^\bullet_{i-1}\), \(i < N\) as shown in the bottom row of the diagram. Note that this sequence starts from \(\mathcal{G} = S\), i.e., sketches with the empty signature, which allows us to defined mapping \(\alpha^\bullet_0\), which in turn defines sketch category \(S_0\) and so on. Bottom arc arrows show the carriers. Diagonal dotted arrows labelled :sig are pairs rather than mappings (and can be seen as instances of some big mapping sig from the big category of all sketches to the big category of all signatures, which we don’t consider in the paper).

The above defines a dh-signature \(\mathcal{P}^\bullet\). Sketched over \(\mathcal{P}^\bullet\) and their morphisms are defined as objects and arrows of the category \(S_{N-1} \overset{\text{def}}{=} S_{N-2}||\mathcal{P}^\bullet_{N-1}\). We will denote this category by \(G||\mathcal{P}^\bullet\): strictly speaking, we should write \(G||^\ast\mathcal{P}^\bullet\) as it’s another construct of building sketches, but we will usually omit the superscript near || if the right term unambiguously shows the dh-context of ||.

**Theorem 2.** Any dh-signature \(\mathcal{P}^\bullet\) gives rise to a functorial signature \(\overline{\mathcal{P}}^\bullet\), and any functorial signature \(\mathcal{P}\) gives rise to a dh-signature \(D^\ast\mathcal{P}\). Moreover, \(\overline{\mathcal{D}^\ast\mathcal{P}} \cong \mathcal{P}\) and \(\mathcal{D}^\ast(\overline{\mathcal{P}^\bullet}) \cong \mathcal{P}^\bullet\).

**Proof.** Let \(\mathcal{P}^\bullet\) be a dh-signature as defined above. Makkai’s theorem allows us to rewrite \(a_2: S_1 \leftarrow \mathcal{P}^\bullet_2\) as \(a_2: \text{Set}^\bullet \leftarrow \mathcal{P}^\bullet_2\), then currying gives us a profunctor

\[
\hat{a}_2: \text{Set} \leftarrow (G \star a_1, \mathcal{P}^\bullet_1) \times \mathcal{P}^\bullet_2
\]

(we don’t write \(\mathcal{P}^\bullet_2\) as \(\mathcal{P}^\bullet_2\) is discrete), whose collage is the category

\[
(G \star a_1, \mathcal{P}^\bullet_1) \star a_2 \mathcal{P}^\bullet_2.
\]

The latter, for any \(P \in \mathcal{P}^\bullet_2\) and \(P' \in \mathcal{P}^\bullet_1\), has a wavy arrow \(\varpi: P \rightarrow P'\) iff there is a diagram index \(c \in \mathcal{P}^\ast\mathcal{P}(P')\) (i.e., an element in \(\mathcal{P}^\ast\mathcal{P}\) for sketch \(\mathcal{P}^\ast\mathcal{P}\)), and similarly, any wavy arrow \(\varpi: P \rightarrow H\) is generated by a cell \(a \in \mathcal{P}^\ast\mathcal{P}(H)\) in the carrier of sketch \(\mathcal{P}^\ast\mathcal{P}\)— we use overlining and underlining as earlier in definitions (10) and (11) (Example in Fig. 9 shows how it works). We continue for \(i = 3\ldots N\) and obtain the commutative diagram formed by the upper lane of squares in (22):

\[
\emptyset \rightarrow \mathcal{P}^\bullet_0 \rightarrow \mathcal{P}^\bullet_1 \rightarrow \ldots \rightarrow \mathcal{P}^\bullet_N
\]
Table of Contents

In this lane, the lower line (starting at $\mathcal{G}$) is a chain of collage categories obtained by inductively adding binding/wavy arrows (taken from the arity sketches elements) layer by layer. (Example in Fig. 9 would illustrate the idea if we make all orange arrows black wavy arrows.) By removal of the subcategory $\mathcal{G}$ (and resp. all arrows from predicates to $\mathcal{G}$-objects) from each of the collages, we obtain the bottom chain of categories $\mathcal{P}_1^*$. (With the colour legend of the example in Fig. 9, we remove all black arrows from predicate symbols to $\mathcal{G}$ but keep arrows between predicates and colour them in orange. Thus, the difference between set $\mathcal{P}_i^*$ and category $\mathcal{P}_i^*$ is the presence of (formerly wavy and black but now) orange arrows $\alpha$ between predicate symbols specified above.) The arrows that left in categories $\mathcal{P}_i^*$ actually specify dependencies, and we can consider category $\mathcal{P}_N^*$ as the carrier of a f-signature with dependencies. To complete the story and make $\mathcal{P}_N^*$ a signature, we need to define the corresponding (integrated) arity functor

$$\{\alpha\colon \mathcal{P}_N^* \to \mathcal{G}^{op}\}$$

(23)

On objects, for any $i \leq N$ and $P \in \mathcal{P}_i^*$, we set $\{\alpha(P)\} \overset{\text{def}}{=} |\alpha_i(P)|$.

For arrows, given a binding/wavy arrow $\alpha\colon P \to P'$ for $P' \in \mathcal{P}_{i-1}^*$, we define

$$\{\alpha(\alpha)\} \overset{\text{def}}{=} \text{diagr}^{\alpha_i}(P)(c): |\alpha_{i-1}(P')| \to |\alpha_i(P)|$$

(24)

Checking the functoriality of so defined $\{\alpha\}$ is straightforward, which makes the pair $(\mathcal{P}_N^*, \{\alpha\})$ a correct f-signature. We will denote it by $\mathcal{F}^*$ (so that now the carrier $\mathcal{P}_N^*$ is also denoted by $\mathcal{F}_N^*$).

Now we consider the transition in the opposite direction. Let $\mathcal{P}$ be an f-signature of height $N$ with arity functor $\alpha\colon \mathcal{P} \to \mathcal{G}^{op}$. Recall that f-signatures are layered, layers are sets, and we define $\mathcal{P}_1^* = \mathcal{P}_i$ for all $1 \leq i \leq N$. For each $P \in \mathcal{P}_i^*$, we define sketch $\alpha_i(P)$ as follows. The carrier $|\alpha_i(P)| = \alpha(P)$.

Diagrams are given by dependency arrows; for any $P' \in \mathcal{P}_{i-1}^* = \mathcal{P}_{i-1}$, we set $\alpha_i(P') = \{p\} p \in \mathcal{P}(P, P')$ (recall that formally $p$ is just a couple $(p, \#)$ of $p$ with any aside token $\#$), and for any index $p \in \{P\}$, the diagram $\text{diagr}(p)$ is defined to be mapping $\alpha(p)$ (the construction is exactly the inverse to (24)). Thus, we have decomposed f-signature into a dh-signature $\mathcal{D}^*\mathcal{P}$, in which dependency arrows are hidden in diagrams of the arity sketches. Isomorphisms immediately follow from the constructions above.

Corollary 5. a) Given an f-signature (i.e., predicate signature with dependencies) $\mathcal{P}$, two categories of sketches, $\mathcal{G}||\mathcal{P}$ and $\mathcal{G}||\mathcal{D}^*\mathcal{P}$ are isomorphic. b) Given a dh-signature $\mathcal{P}^*$, two categories of sketches, $\mathcal{G}||\mathcal{P}^*$ and $\mathcal{G}||\mathcal{F}^*$, are isomorphic as well.7

4.5 Some operations on sketch subobjects

Definition 15 (Operations between subobject semi-lattices). Let $\mathcal{P}$ be a predicate signature and $S \in \mathcal{G}||\mathcal{P}$ a sketch. There is a subobject functor

$$\text{subsk}_S : \text{Sub}(|S|) \to \text{Sub}(S)$$

7 The bulky formulation of this theorem should be shorter and properly stated if we introduce categories of f-signatures and dh-signatures, the corresponding categories of sketches over them, and two functorial isomorphisms. We leave it for future work.
which adds to a subgraph $G \subseteq |S|$ all $S$-constraint that can be declared on $G$. It is defined via the right adjoint $\forall (id, !)$ of the pullback functor $(id, !)^{-1}: \text{Sub}(S) \rightarrow \text{Sub}(|S|)$, see Remark ???. Note that $|S|$ can be interpreted as a sketch without diagrams, thus there is a canonical sketch morphism $(id, !): |S| \rightarrow S$ in $\mathcal{G}||\mathcal{P}$.

Furthermore, there is another functor

\[ \text{subsig}_\mathcal{P}: \mathcal{G}||\mathcal{P} \rightarrow \text{Sub}(\mathcal{P}) \]

which maps a $\mathcal{P}$-sketch $S$ to its effective predicate signature, i.e. those predicates from $\mathcal{P}$ that are actually used in the sketch, $\text{subsig}_\mathcal{P}(S) \subseteq \mathcal{P}$. It is defined as the image of $\text{label}$, i.e. $\text{subsig}_\mathcal{P}(S) = \text{Img}(\text{label})$. Arrows in $\text{Sub}(\mathcal{P})$ are inclusions and $\text{subsig}_\mathcal{P}$ is obviously a functor.

Finally, if $\mathcal{P}'$ is a subsignature of $\mathcal{P}$, i.e., $|\mathcal{P}'| \subseteq |\mathcal{P}|$ in $\text{Cat}$, and $\text{proj}_\mathcal{P} = \alpha$, then we have a forgetful functor $\text{proj}_{\mathcal{P}'}: \mathcal{G}||\mathcal{P} \rightarrow \mathcal{G}||\mathcal{P}'$.

The rest of the page is intentionally empty.
5 Basic Diagram Algebra: Operations without Dependencies

In this section we consider operation signatures and algebras for them when there are no dependencies between operations. Syntax and semantics of the parallel composition of queries (we say multiqueries) are described in, resp., Sect. 5.1 and 5.2, and sequential composition (complex queries) is in Sect. 5.3. In Sect. 5.4 we define views and their compositions, and show that they can be organized into an sm-category. Section 5.5 discusses how our query building mechanism is related to monads.

Our technical machinery in this section is basically a “pushout calculus”: we chase PO squares and wide POs over categories of sketches (which are presheaf toposes by the extended Makkai’s theorem). All these sketch categories are assumed to be skeletal by default, which makes POs over them deterministic operations. The skeletality assumption is motivated by our applications as discussed in Sect. 2.2. However, while applications need finitary operations, until sect. 5.3 on complex queries, we consider the general case without the finitarity assumption as it does not bring any additional technical difficulties (and notationally it is even simpler to work with indexing in general, say, \(x_i, i \in I\), rather than with finite enumerations \(i = 1, 2, ..., n\)). Moreover, although complex queries are finite sequences of multiqueries, the latter are not assumed to be finitary until Sect. 5.5, where size restriction become important for building the query monad, and we prove that complex queries built from finitary multiqueries give rise to a finitary monad.

5.1 Multiqueries and Operations over them

Basic Notions. Recall Def. 3 on p. 15 of an operation signature. We now consider how we can build complex operations (queries) from elementary ones given in the signature. We first need the notion of an atomic query that makes a derived operation from a single operation, e.g., \(x \cdot x\) makes a unary operation from a binary one (and later we will consider multi-queries).

Definition 16 (Atomic Queries (= atomic terms)).
Let \(\mathcal{O}\) be a discrete operation signature over a category of sketches \(S = G|\mathcal{P}\) (see Def. 3). An atomic query is a triple \(q = (I_q, \phi, b_q)\) of an input sketch \(I_q \in S\), an operation symbol \(\phi \in \mathcal{O}\) (we omit the subscript \(q\)) and a binding sketch morphism \(b_q : I_\phi \to I_q\). These data determine pushout squares \((25)\) in \(S\) (where we write \(IE_\phi\) for \(\alpha(\phi)\)), in which the resulting objects \(E_q\) is determined up to iso.

\[
\begin{array}{ccc}
I_\phi & \xrightarrow{IE_\phi} & E_\phi \\
\downarrow b_q & & \downarrow b_q^E \\
I_q & \xrightarrow{IE_q} & E_q
\end{array}
\]

(25)

An atomic query is finitary if \(\phi\) is finitary, i.e., sketch \(E_\phi\) (and hence \(I_\phi\)) is finite.

Remark 2 (Concrete Syntax). By fixing a concrete syntax for specifying terms, we fix a particular object \(E_q\), while the general pushout formulation gives us a concrete-syntax-free description. Consider, for example, a binary operation + in the ordinary universal algebra setting (Example 3) and an atomic query \(q = (X, +, b_q)\) over a set of variables \(X = \{x, y, z\}\) in the role of \(I_q\) and a binding mapping \(b_q : 2 \to X\), where \(2 = \{0, 1\}\) and \(b_q = \{0 \rightarrow x, 1 \rightarrow y\}\). The diagrammatic arity of + is an inclusion \(IE_+: \{0, 1\} \to \{0, 1, \ast\}\) with \(\ast\) being an arbitrarily

McSCert report 2021-34 32 May 11, 2021
chosen token, and taking pushout of \((IE_+, b_\oplus)\) gives us a four-element set, e.g.,

\[ E = \{ x, y, @, z \} \] with \(@\) being a new element representing the application of \(+\) to \((x, y)\) (indeed, \(b_\oplus^x(\oplus) = @\) irrespective of what tokens are chosen for \(\ast\) and \(\oplus\)).

If elements of set \(X\) are chosen to be strings (a typical case), then for the new element we may take a string \(x + y\) (infix) or \(+ (x, y)\) (prefix) or \(xy +\) (reversed polish notation) etc. depending on the respective choice of the concrete syntax. Thus, choosing a concrete syntax for an atomic term \(q\) amounts to choosing a concrete pushout square for the set of binding squares determined by \(q\). However, as we will not use any special properties of a chosen pushout square besides its universality, our considerations are actually concrete-syntax-free, i.e., do not depend on the chosen concrete syntax. Thus, working with chosen POs would mean working with some chosen but unspecified concrete syntax (analogously to defining a function with an unspecified parameter). However, we work with an even stronger skeletonality assumption, which makes our PO constructions deterministic and analogous to classical ASTs (abstract syntax trees) in the ordinary algebra.

An algebraic analog of the notion of sketch (a conjunctive theory) is the notion of multiquery—a “conjunction” of atomic queries executable in parallel.

**Definition 17 (Multiqueries).** A multiquery \(Q\) over a discrete one-layer signature \(\mathcal{O}\) (over a category of sketches \(S = \mathbb{G} | \mathbb{P}\)) is given by a sketch \(I_Q \in S\) called the input, and a family of atomic queries with the same input sketch \(I_Q\). This family is given by a span of two functions \(\text{label}\) and \(\text{diagr}\) making a commutative diagram (26b) in \(\text{Set}\). The apex of the span is a set \(\text{Index}(Q)\) of multiquery indexes, and for an index \(q \in \text{Index}(Q)\), diagram \(\text{diagr}(q)\) or \(\text{diagr}^q\) is a pushout square in \(S\) of shape shown in (26a) in which the value \(\text{diagr}^q(I_\oplus)\) is predefined to be \(I_Q\). Moreover, \(\text{diagr}^q(\mathbb{G}) = (\text{label}^q)^\alpha\) due to commutativity, and hence the triple \((q, \text{label}^q, \text{diagr}^q(I_I))\) is an atomic query.

![Diagram](image)

**The image of function label, the set \(\{\text{label}(q) | q \in \text{Index}(Q)\}\) of all operation symbols involved in \(Q\) (together with their arities) will be denoted by \(\mathcal{O}(Q) \subset \mathcal{O}\).**

The internal square in (26b) is a pullback in \(\text{Set}\), and hence there is a unique mapping \# . Note that what the pull-back produces is nothing but the set of all
(diagrammatic) terms in Cartesian sense, i.e., without sharing, and mapping \# is thus an important part of the picture, which says that computationally different terms can denote the same universal-algebraic term (this is discussed i) and shows the difference between a combinatorial notion of a term and the target of this mapping is nothing but a A multiquery is mono, if the pair (label, diagr) is jointly monic and hence \# is monic. A multiquery \(Q\) is finitary if all operations \(\text{label}(q), q \in \text{Index}(Q)\) are finitary, and \(Q\) is finite if the indexing set \(\text{Index}(Q)\) is finite.

**Example 6 (Running Example).** In Fig. 3 on p. 6, the two applications \(a, b\) of the pullback operation (see two chevrons in diagram B) form a multi-query \(Q\) with \(\text{Index}(Q) = \{a, b\}\), \(\text{label}(a) = \text{label}(b) = [\text{pb}]\) and the following binding diagrams (see Fig. 8(C) on p.20 for naming the elements in the arity of [pb]):

\[
\text{diagr}(a)(II) = \{10 \mapsto x, 20 \mapsto y\}; \quad \text{diagr}(a)(EE) = \{10 \mapsto x, 20 \mapsto y, 32 \mapsto x_y, 31 \mapsto y_x\},
\]

\[
\text{diagr}(b)(II) = \{10 \mapsto z, 20 \mapsto y\}, \quad \text{diagr}(b)(EE) = \{10 \mapsto z, 20 \mapsto y, 32 \mapsto z_y, 31 \mapsto z_x\},
\]

and the mappings for horizontal monics are easily restored. The result of applying this multiquery is a black-and-blue ([monic]-sketch in Fig. 3(B) on p. 6.

We can use multi-queries to extend our operation signature in the same way as we can define new methods in a programming language by combining built-in methods. For example, the multiquery above is an operation that takes diagram \(I = (x, y, z)\) as an input and completes it with arrows \(x_y, y_z, z_y\) as shown in the figure. On the other hand, multiqueries can be seen as sketches in an extended signature. For example, the multiquery above has two chevron labels of the square arity \(E_{[\text{pb}]}\) (the scope of operation [pb]) and, implicitly, two labels of the cospan arity \(I_{[\text{pb}]}\) that describe the inputs to [pb] applications. Moreover, there is a dependency between the corresponding operation symbols \(P^E_{[\text{pb}]}\) and \(P^l_{[\text{pb}]}\); as soon as a diagram labelled \(P^E_{[\text{pb}]}\) is found in a sketch, the corresponding diagram \(P^l_{[\text{pb}]}\) must be there too. The following definition makes it precise.

**Example 7 (Empty multiqueries).** For any sketch \(X\) with \(\text{Index}(0_X) = \emptyset\) and empty leg mappings too.

**Operations over Multiqueries.** We begin with an important construction of extended signature.

**Definition 18 (Extended Signature).** For a given predicate signature \((\mathcal{P}, \alpha_{\mathcal{P}})\) and an operation signature \((\mathcal{O}, \alpha_{\mathcal{O}})\) over \(\mathcal{P}\), their collage signature is a predicate signature \((\mathcal{P}||\mathcal{O}_E, \alpha_{||})\) with \(\text{Ob}(\mathcal{P}||\mathcal{O}_E) = \text{Ob}(\mathcal{P}) \sqcup \mathcal{O}\) and \(\text{Arr}(\mathcal{P}||\mathcal{O}_E) = \text{Arr}(\mathcal{P})\) so that \(\mathcal{P} \subset \mathcal{P}||\mathcal{O}_E\); further, \(\text{proj}_{\mathcal{P}} = \alpha_{\mathcal{P}}\) and the object part of \(\alpha_{||}\) is provided by the following diagram:

\[
\begin{array}{ccc}
\mathcal{P} & \xrightarrow{\alpha_{\mathcal{P}}} & \text{Ob}(\mathcal{P}) \sqcup \mathcal{O} \xrightarrow{\alpha_{\mathcal{O}}} \mathcal{O} \\
\text{Ob}(\mathcal{G}) & \xrightarrow{\text{Ob}([\cdot\mapsto, \mathcal{G}||\mathcal{P}])} & \text{Ob}(\mathcal{G}||\mathcal{P}) \\
\mathcal{G} & \xrightarrow{\text{Ob}(\mathcal{G})\text{cod}} & \text{Ob}(\mathcal{G}||\mathcal{P})
\end{array}
\]

The sketch category \(\mathcal{G}||\mathcal{P}||\mathcal{O}_E\) will be denoted by \(\mathcal{G}||\mathcal{P}||\mathcal{O}_E\) or \(\mathcal{S}||\mathcal{O}_E\). Any sketch \(S \in \mathcal{G}||\mathcal{P}||\mathcal{O}_E\) can be seen as a \(\mathcal{P}\)-sketch \(\text{proj}_{\mathcal{P}}\) by forgetting its \(\mathcal{O}\)-diagrams.
Lemma 1 (Multi-queries as operation symbols) Any multi-query $Q$ with input sketch $X \in S := \mathbb{G}||\mathbb{P}$, gives rise to a $\mathbb{P}||\mathbb{O}_E$-sketch $Q(X)$ and a monic

$$\{ \eta_Q : X \hookrightarrow Q(X) \} \in \text{Arr}(\mathbb{G}||\mathbb{P}||\mathbb{O}_E).$$

Restriction $\text{proj}_P$, will be denoted by $E_Q$ and we thus also have a $\mathbb{P}$-sketch monic

$$\{ IE_Q : X \to E_Q \} \in \text{Arr}(S)$$

called the arity monic of $Q$ ($\eta_Q$ is also re-denoted as $IE_Q$; the change in notation will become clear after Lemma 3, in which we show that each $\mathbb{O}$-algebra $A$ is also a $IE_Q$-algebra).

Proof. We will first build monic $IE_Q : X \to E_Q$ in $S$, and then will show $E_Q$'s enrichment with $\mathbb{O}$-diagrams. We take projection

$$\text{proj}_{IE_Q} : [X \to \cdot, S] \leftarrow \left[ \mathbb{G}[\eta_Q], S \right]$$

and obtain

$$\text{diagr'} \overset{\text{def}}{=} \text{diagr} \circ \text{proj}_{IE_Q} : \text{Index}(Q) \to [X \to \cdot, S]$$

Currying gives us a wide span with apex $X$

$$\text{diagr'} : \text{Index}(Q) \times \{ X \to \cdot \} \to S$$

whose colimit (wide pushout) in $S$ gives us a monic $IE_Q : X \to E_Q$ together with a family of monics $(EE^q : E^q \to E_Q \mid q \in \text{Index}(Q))$ as shown in the diagram below for the case of a countable $\text{Index}(Q)$ (think of a countable pile of binding squares glued together at $I_Q$, and the colimit cocone is shown with dashed arrows):

$$\text{diagr'} : \text{Index}(Q) \times \{ X \to \cdot \} \to S$$

whose colimit (wide pushout) in $S$ gives us a monic $IE_Q : X \to E_Q$ together with a family of monics $(EE^q : E^q \to E_Q \mid q \in \text{Index}(Q))$ as shown in the diagram below for the case of a countable $\text{Index}(Q)$ (think of a countable pile of binding squares glued together at $I_Q$, and the colimit cocone is shown with dashed arrows):

In addition, we have a family of diagrams indexed by $q \in \text{Index}(Q)$:

$$b^q_E \triangleright EE^q : E_{\text{label}(q)} \to E_Q,$$

which makes $E_Q$ a $G||P||O_E$-sketch $Q(X)$ or just $Q(X)$. Figure 10 presents the construction above in a compact way as a metasketch.

Corollary 6 (Empty Queries as Identities). For any sketch $X$, the arity of the empty query over $X$ is identity: $\eta_{0_X} = \text{id}_X : X \to X$. 

McSCert report 2021-34 35 May 11, 2021
Multiquery Merge

**Definition 19 (Multiquery Morphisms).** Let $X$ be a sketch in $S = \mathcal{G}||\mathcal{P}$, and $Q, Q'$ are two multiqueries over $X$ (see Def. 17). A morphism $f: Q \rightarrow Q'$ is a function $f: \text{Index}(Q) \rightarrow \text{Index}(Q')$ such that $f : \text{label} = \text{label}'$ and $f : \text{diagr}' \triangleright \text{proj}^{IE} = \text{diagr} \triangleright \text{proj}^{IE}$. Clearly, due to commutativity, we also have $f : \text{diagr}' \triangleright \text{proj}^{IE} = \text{diagr} \triangleright \text{proj}^{IE}$ and hence, as pushouts are unique for sketch categories, $f : \text{diagr}' = \text{diagr}$. Hence, multiquery morphism is nothing but a span morphism.

**Corollary 7.** A multiquery morphism $f: Q \rightarrow Q'$ is also a $\mathcal{G}||\mathcal{P}||\mathcal{O}_E$-sketch morphism making the following diagram commutative:

\[
\begin{array}{ccc}
X & \xrightarrow{\eta_Q} & Q(X) \\
\downarrow & & \downarrow f \\
Q(X) & \xrightarrow{\eta_{Q'}} & Q'(X)
\end{array}
\]

**Proposition 3.** For any $\mathcal{P}$-sketch $X$, there is a (big) category of multiqueries over $X$, $\mathcal{MQ}(X)$, equipped with an injective on objects and faithful functor $\text{mq2sk}$ as shown below:

\[
\begin{array}{c}
\mathcal{MQ}(X) \\
\xrightarrow{\text{mq2sk}} \\
\mathcal{G}||\mathcal{P}||\mathcal{O}_E \\
\xrightarrow{\text{proj}} \\
\mathcal{G}||\mathcal{P}
\end{array}
\]

We will also write $m\mathcal{QQ}(X)$ for the class $\text{Ob}(\mathcal{MQ}(X))$.

**Proof.** Obviously, multiquery morphism composition is (strictly) associative, and the empty multiquery $0_X$ is the unit. This gives us category $\mathcal{MQ}(X)$. We will
consider its (co)skeletal version, in which isomorphic multiqueries are identified. Injectivity and faithfulness of \(mq2sk\) follow from Def. 17 and 19. The functor is not full, e.g., there are sketch morphisms whose carrier graph part is an endomorphism but multiquery morphisms assume this part to be identity on \(X\).

**Proposition 4 (Multiquery Merge).** For any \(X\), category \(\mathbb{MQ}(X)\) has coproducts (denoted by \(\mathbb{MQ}(X)\) or \(+_X\) or just \(+\) if the context is clear). Moreover, pushout of span \((\eta_Q, \eta_{Q'})\) in \(G||P||O_E\) (see diagram (29a)) produces \((Q + Q')\) with legs being coprojections: \(\eta^*_{Q'} = \iota : Q \rightarrow Q + Q'\) and \(\eta^*_{Q} = \iota' : Q' \rightarrow Q + Q'\) (note switching the primes in our notation).

**Proof.** Let \(Q\) and \(Q'\) be two multi-queries over \(X\). We construct a multi-query \(Q + Q'\) called the merge in the following way. We define the apex set

\[
\text{Index}(Q +_{\mathbb{MQ}(X)} Q') := \text{Index}(Q) \cup \text{Set} \text{Index}(Q'),
\]

and then the legs label \(\eta^+\) and diag \(\eta^+\) are determined by the universal property of the coproduct in \(\text{Set}\): the required commutativity of the rectangle is straightforward. Checking the universality of \(+_{\mathbb{MQ}(X)}\) is straightforward.

Now we prove that diagram (29a) in \(G||P||O_E\) is a pushout. We use construction of Lemma 1 and note that the wide span in diagram (27) for \(+_{\mathbb{MQ}(X)}\) \(Q'\) is partitioned into two wide spans, and hence its colimit is given by the pushout diagram of subspans colimits, but the latter are specified by the two \(\eta\)-arrows. Diagram (b) shows the \(\text{proj}\) projection of (a) (note that \(X\) is a \(P\)-sketch).

\[
\begin{array}{ccc}
Q(X) & \xrightarrow{\eta_Q} & X \\
\downarrow{\eta^*_{Q'}} & & \downarrow{\eta^*_{Q}} \\
(Q +_{\mathbb{MQ}(X)} Q')(X) & \xrightarrow{\iota} & Q'(X)
\end{array}
\quad
\begin{array}{ccc}
X & \xrightarrow{\iota} & Q(X) \\
\downarrow{\iota'} & & \downarrow{\iota} \\
(Q +_{\mathbb{MQ}(X)} Q')(X) & \xrightarrow{\iota Q} & Q'(X)
\end{array}
\]

(a) PO in \(G||P||O_E\) \quad (b) PO in \(G||P\)

**Multiquery Substitution** In universal algebra, we can freely exchange the variable names in a term algebra \(T(X)\). Thus every variable substitution \(\sigma : X \rightarrow Y\) freely induces a unique term-algebra substitution \(T(\sigma) : T(X) \rightarrow T(Y)\). The same is true for our notion of multiqueries.

**Lemma 2 (Multiquery Substitution)** Any \(P\)-sketch morphism \(f : X \rightarrow X'\) gives rise to a functor \(f_{\mathbb{MQ}} : \mathbb{MQ}(X) \rightarrow \mathbb{MQ}(X')\). Moreover, for any \(m\)-query \(Q \in \mathbb{MQ}(X)\), we have a pushout square in \(G||P||O_E\) shown in diagram (30)(b) below, where \(Q^f\) denotes \(f_{\mathbb{MQ}}(Q)\) and \(f_\sigma^*\) is the lifting of \(f\) at \(Q\) (we will show later that it is indeed an opcartesian lifting). Diagram (c) is merely another notation for diagram (b), which may be useful in some cases, e.g., for stating compositionality of the construction accurately specified in diagram (d), but in terms of (c) taking the form \(Q(f \circ g) = Q(f) \circ Q(g)\). Diagram (a) is the \(\text{proj}\) projection of (b), and (e) will be used in the proof.
Proof. We define multiquery query $Q^f = (I_{Q^f}, label^f, diagr^f)$ as follows. The input sketch is $I_{Q^f} := Y$ and the indexing span is

$$\mathcal{O} \xrightarrow{\text{Index}(Q^f)} diagr^f \xrightarrow{\text{proj}_{IE}} Q^f$$

where $Index(Q^f) = Index(Q)$ and $label^f = label$ are taken from multiquery $Q$. Functor $diagr^f$ is defined as follows. For each $q \in Index(Q)$, we define the diagram $diagr^f(q)$ by setting:

(a) $diagr^f(q)(IE) = diagr(q)(IE) = IE_{label(q)}$;

(b) $diagr^f(q)(II) = diagr(q)(II) \circ f = b_{q} \circ f$;

(c) arrows $diagr^f(q)(IE_{q})$ and $diagr^f(q)(EE)$ are given by the pushout of the two arrows above, which can be decomposed into two pushouts (1) and (2) as shown in diagram (e) (note also that monomorphisms are preserved by pushouts since $G||P$ is adhesive). We can thus say that diagram (e) gives us substitution for each atomic query $q \in Index(Q^f)$, and query $Q^f$ is defined. Commutativity of the square $label^f \circ \alpha = diagr^f \circ \text{proj}_{IE}$ is obvious by construction.

By Lemma 1 both multiqueries $Q$ and $Q^f$ have their arity monics $IE_Q$ and $IE_{Q^f}$ (see diagram (32) hosted by category $G||P$) computed by the respective colimits; in the diagram, the upper face is the colimit cocone for $Q$ and the bottom face is the colimit cocone for $Q^f$. Squares $I_Q E_{q}, E_{q^f} Y$ are pushout squares.
Universality of the upper colimit gives us arrow $f^*_Q$, but we need to prove that the face square $I_QE_QE_QfY$ is a PO. We prove it by a direct demonstration that the square is universal. Let $(x, y)$ be a cospan making a commutative square with $IE_Q$ and $f$. Then for each index $q$ (shown in the diagram as $q_i$), as the square $I_QE_QE_QfY$ is a pushout, we have a unique arrow $u_q$ (shown in the diagram as $u_i$). The family of these $u_q$ makes a cocone over $IE^q$, and as $E_Q^q$ is the colimit cocone, there is a unique arrow $u!$ making all diagrams commutative. Hence, the front face is a pushout, and we built diagram (30)(a).

It is easy to see that this diagram is also a pushout in $G\|P\|O_E$. Recall from Lemma 1 that both multiqueries provide also monics $\eta_Q$, $\eta_Q^f$ in $G\|P\|O_E$, and as label$^f$ = label by construction, mapping $f^*_Q$ is a correct $G\|P\|O_E$-morphism. Now, if arrow $x$ in the diagram above is a $G\|P\|O_E$-morphism, then $G\|P\|O_E$-arrow $u!$ is automatically a correct $G\|P\|O_E$-morphism, which gives us diagram (30b). The functoriality of the construction immediately follows from the pushout composition lemma applied to diagram (30d).

Now suppose that $r: Q \rightarrow Q'$ is a m-query morphism. We extend diagram (30b) to a triangle prism as follows. The upper face is given by the triangle diagram in Corollary 7 (on p. 36), and then we take PO of $(f, \eta_Q^f)$ by applying PO decomposition to $\eta_Q^f = \eta_Q | r$ which completes the prism. The right face is a PO square $(r, f_Q, f^*_Q, r^*)$ and we define $f^{MQ}(r) = r^*$. If $r': Q' \rightarrow Q''$ is
another m-query morphism, then PO decomposition applied again gives us required functoriality \( f^{MQ}(r \downarrow r') = f^{MQ}(r) \downarrow f^{MQ}(r') \) — this is a horizontal version of diagram (30d).

The Grothendieck construction gives us the following

**Corollary 8.** The multiquery construction gives rise to an indexed category functor \( \tilde{MQ}: \mathcal{G}\|\mathcal{P} \rightarrow \mathcal{Cat} \) or, equivalently, split opfibration \( \mathcal{G}\|\mathcal{P} \leftarrow \tilde{MQ} \), where \( \tilde{MQ} \) is the total category of all multiqueries. Moreover, reindexing is compatible with multiquery merge.

Merge preservation easily follows from the construction considered in the proof of Lemma 2.

5.2 Multiquery Semantics

The next step is to show that any \( \mathcal{O} \)-algebra is also a carrier of the \( Q \)-operation of arity \( IE_Q \). We will begin with an accurate re-definition of the notion of an \( \mathcal{O} \)-algebra.

**Definition 20 (Algebras (redefined)).** An algebra \( \mathcal{A} \) over \( \mathcal{O} \) is given by a carrier \( |\mathcal{A}| \) and a mapping \( [\_]^A \), shown in diagram (33b), where the span \( (\alpha'_A, \text{proj}_{IE}) \) is derived as a pullback in \( \mathcal{S} \). Elements in its apex \( \text{LDiagr}(\mathcal{A}) \) can be identified with pairs \( \psi, \delta \) in which \( \delta \) is a pushout diagram in \( \mathcal{S} \) with the left-bottom corner being \( |\mathcal{A}| \) and the upper monic being \( \phi^\alpha \) due to commutativity.

Function \( [\_]^A \) maps any such pair to a triangle diagram in \( \mathcal{S} \), in which

- the left-side (binding) arrow is the same as in diagram \( \delta \) due to the commutativity of the trapezoid (a),
- the upper monic is \( \phi^\alpha \) due to commutativity \( \alpha'_A \uplus \text{proj}_{IE} = [\_]^A \uplus \text{proj}_{IE} \);
- hence, the right-side arrow can be considered as the result of the operation: in fact, the right-side “formal” arrow in the pushout square \( \delta \) freely added to graph \( |\mathcal{A}| \) is replaced by an actual arrow in \( |\mathcal{A}| \), namely, \( \text{proj}_{EE} \).

Thus, diagrams at the target of arrow \( [\_]^A \) are actually commutative triangles from Def. 2.
Lemma 3 (Multiqueries as operations) Let \( Q \) be a multiquery in a discrete signature \( \mathcal{O} \). Then any \( \mathcal{O} \)-algebra is also an \( \mathcal{O} + Q \) - algebra\(^8\), i.e. any binding \( b: I_Q \to |A| \) gives rise to a unique \( [Q]^A(b) \): \( E_Q \to |A| \) such that \( IE_Q \downarrow [Q]^A(b) = b \).

Instead of saying that \( A \) is a \( \mathcal{O} + Q \)-algebra we may simply say \( A \) is a \( Q \)-algebra when \( \mathcal{O} \) is clear from the context.

Proof. In diagram (33), we add to the internal rectangle defining an algebra \( A \), the external rectangle defining a multiquery \( Q \). Let \( b: I_Q \to |A| \) be a sketch morphism. Then post-composition gives us arrow \( \downarrow b \) as shown in the diagram, and universality of pullbacks gives us arrow \( !_b \). Composing the latter with \( [.]^A \) gives us evaluation of all atomic queries in \( Q \) and a wide cospan \( (EA_q; E_q \to A \sqcap q \in \text{Index}(Q)) \), where \( EA_q = \text{proj}_E \). Now diagram (28) can be completed as shown with dashed lines in diagram (34) below for the case of finite \( \text{Index}(Q) \), and the universality of the colimit gives us a required arrow \( ! \) such that \( b = IE_Q \downarrow ! \). That is, we have defined an operation \( [Q]^A \) that maps any input binding \( b \) to a scope binding \( b^E = ! \).

5.3 Complex Queries

Multiqueries can be sequentially composed into complex queries.

Definition 21 (Complex queries). Let \( X \) be a sketch in \( \mathbb{G}\| \mathcal{P} \). A complex query \( Q \) over \( X \) is a finite sequence of multiqueries \( \{Q_1, \ldots, Q_{N(Q)}\} \) such that \( I_{Q_1} = X \) and \( I_{Q_{i+1}} = E_{Q_i} \), for all \( i \leq N(Q) \). The number \( N(Q) \) is the length of the query, and we will simply write \( N \) if the query \( Q \) is clear. Thus, we have a sequence of monics in \( \mathbb{G}\| \mathcal{P}\| \mathcal{O}_E \):

\[
I_Q = X_0 \xrightarrow{\eta_1} X_1 \xrightarrow{\eta_2} \cdots X_{N-1} \xrightarrow{\eta_N} X_N = Q(X)
\]

where \( X_i := Q_i(X_{i-1}) \), \( i \leq N \); its composition will be denoted by \( \eta_Q: X \to Q(X) \).

We also have the projection of the above to \( \mathbb{G}\| \mathcal{P} \):

\[
I_Q = I_{Q_1} \xrightarrow{IE_{Q_1}} E_{Q_1} = I_{Q_2}, \ldots, E_{Q_{N-1}} = I_{Q_N} \xrightarrow{IE_{Q_N}} E_{Q_N} = E_Q
\]

\(^8\) where \( |\mathcal{O} + Q| = |\mathcal{O}| \sqcup |Q| \) and \( \text{proj}_\mathcal{O} = \alpha^{\mathcal{O}}, \alpha^{\mathcal{O}+Q}(Q) = IE_Q: I_Q \to E_Q \)
whose composition will be denoted by $IE_Q$.

Note that an multi-query $Q$ over sketch $X$ can be interpreted as a complex query $\eta_Q: X \rightarrow Q(X)$ of length 1. And empty multiquery $0_X$ is a complex query of length 0. Below we will often write $m$-query and $c$-query for, resp., multiqueries and complex queries.

Two $c$-queries over $X$ of equal length, $Q = (Q_1, ..., Q_N)$ and $Q' = (Q'_1, ..., Q'_N)$, are called equivalent, $Q \cong Q'$, if there is a sequence of multiquery isomorphisms $\iota_i: Q_i \rightarrow Q'_i$ in $MQ(X_i)$, $i = 1, ..., N$ such that $\eta_i \circ \iota_i = \iota_{i-1} \circ \eta'_i$ and $\iota_0 = id_X$.

The class of all $c$-queries over $X$ modulo $\cong$ will be denoted by $cQQ_X$ (and similarly, $\text{Ob}(MQ)$ can be referred to as $mQQ(X)$).

![Complex Query Definition](Fig.11)

Fig. 11 shows a diagrammatic definition of a (complex) query $Q$ as a sequence (possibly infinite) of $m$-queries. Indeed, function $\text{hei}: \text{Index}(Q) \rightarrow \text{Int}$ provides partitioning the set $\text{Index}(Q)$ into disjoint layers (as $i$ is monic and monics are stable under PB), and for any $i > 0$, the corresponding layer gives us a multiquery $Q_i$ due to commutativity of the right upper triangle. Note that in contrast to the multiquery definition (26), the left bottom node in the diagram shape (denoted by bullet) is not fixed but is subjected to a constraint: the right-most up-arrow gives us the sketch substituted for this node, and commutativity of the triangle implies that if for two indexes $q$ and $q$' in $\text{Index}(Q)$, we have $\text{hei}(q) = \text{hei}(q')$, then both diagrams $\text{diagr}(q)$ and $\text{diagr}(q')$ have the same $\bullet$-sketch as their input node. Hence, $q$ and $q'$ can be included in the same multiquery and thus diagram gives us a sequence of multiqueries $Q_i$ with index...
sets $Index(Q_i) = \text{heir}^{-1}(i)$ and the input sketches $I_{Q_i}$. We define $I_{Q_0}$ to be a singleton returning the input sketch $I_Q$ of the complex query $Q$ at hand, i.e., we set $E_{Q_0} = I_Q$. Moreover, there is a unique mapping $\text{heir}$: $\text{Term}(X) \rightarrow \text{Int}$ giving the height of a term. Then arrow $\#$ again glues together computationally different representations of denotationally the same term.

**Definition 22 (Sequential Composition).** Let $Q = (Q_1, \ldots, Q_N)$ and $Q' = (Q'_1, \ldots, Q'_{N'})$ be two complex queries with units $\eta_Q: X \mapsto Q(X)$ and $\eta_{Q'}: E_Q \mapsto Q'(X)$. Then we define a complex query $Q \triangleright Q'$ of length $N + N'$ by the concatenation of sequences $Q$ and $Q'$:

$$Q \triangleright Q' = (Q_1, \ldots, Q_N, Q'_1, \ldots, Q'_{N'})$$

with $\eta_{Q \triangleright Q'} = \eta_Q \triangleright \eta_{Q'}$.

**Lemma 4 (Parallel Composition)** Let $Q = [Q_1, \ldots, Q_N]$ and $Q' = [Q'_1, \ldots, Q'_{N'}]$ be two complex queries over sketch $X \in S$ with $\eta_i: X \mapsto Q(X)$ and $\eta'_i: X \mapsto Q'(X)$ being their respective units. Then there exists a complex query $Q \perp X Q'$ written also as $Q + Q'$ or else very briefly as $Q_+$ with unit $\eta_+: X \mapsto Q_+(X)$ given by the following pushout diagram

![Diagram](image-url)

**Proof.** Suppose first that $N = N'$ and consider the matrix of pushout squares in (36) below built inductively in the following way. We begin with merging queries $Q_i$ and $Q'_i$ over $X$ (by Prop. 4) and obtain diagonal pushout square (11) with monic $\eta^+_i$ denoting $\eta_{Q_i \perp Q'_i}$. Then we apply m-query substitution Lemma 2 to query $Q_2$ over $Q_1(X_0)$ along mapping $\eta^+_1$ and obtain query $Q'_2$ (not shown in the diagram) with pushout square (12) and specifically monic $\eta^+_2$. Similarly, we obtain query $Q'^*_2$ (not shown) with square (21) and monic $\eta'^*_2$ by applying substitution along $\eta^*_1$ to query $Q'_2$. Now we have two m-queries, $Q^*_2$ and $Q'^*_2$ over the same sketch $X_1$ with $X$, which can be merged and produce monic $\eta^*_2 := \eta_{Q^*_2 \perp Q'^*_2}$, and so on until we get the last m-query $Q^*_N + Q'^*_N$ with unit $\eta^+_1 \triangleright \ldots \triangleright \eta^+_N$: $X \mapsto X_N$.

If $N \neq N'$ we extend the shorter query (say, $Q'$) with empty multiqueries so that $Q'' = (Q'_1, \ldots, Q'_{N'}, 0_{N'+1}, \ldots, 0_N)$ and define $Q + Q' := Q + Q''$. 
**Table of Contents**

- Table of Contents
- The proof consists of iterated applications of Lemma 2 to the sequence of m-queries $Q_1, \ldots, Q_n$, which results in the sequence of substituted m-queries $Q'_1, \ldots, Q'_n$, together with respective pushout squares depicted in (39a). The pushout pasting lemma allows us to combine all these pushouts into a big pushout. We can apply the same principle not only horizontally but also vertically.

---

Corollary 9 (Interchange law for complex queries). Let $X$ be a $P$-sketch, and $Q_1, Q_2, Q_3$ and $Q_4$ be c-queries with $X = IQ_1 = IQ_2, IQ_3 = Q_1(X)$, and $IQ_4 = Q_2(X)$. Then the interchange law holds:

$$Q_{1+2} \circ Q_{3+4}(X) = (Q_{1+3} + Q_{2+4})(X)$$

Proof. We take the left part, apply the matrix from Lemma 4 to $Q_{1+3}$ and to $Q_{2+4}$ separately, which will give us two matrices of size $N_{12} \times N_{12}$ and $N_{34} \times N_{34}$ where $N_{12} = \text{max}\{N(Q_1), N(Q_2)\}$ and $N_{34} = \text{max}\{N(Q_3), N(Q_4)\}$ adjoint at the vertex $N_{12}$. Then we apply pushout decomposition lemma to fill-in the empty spaces and obtain a complete pushout matrix of size $(N_{12} + N_{34}) \times (N_{12} + N_{34})$, which gives us the right side of (37).

Lemma 5 (Complex Query Substitution) Let $Q = [Q_1, \ldots, Q_n]$ be a complex query with input sketch $IQ = X$, and $f: IQ \rightarrow Y$ a sketch morphism. These data uniquely determine a complex query $Q^f$ with input sketch $IQ^f = Y$ and unit $\eta_{Q^f}$ given by pushout diagrams (b) in (38) (similar to that in Lemma 2 on p.37) and (a,c) similar to those (a,c) Likewise, $Q$ is functorial: $Q(f \circ g) = Q(f) \circ Q(g)$.

$$I_{Q^f} \xrightarrow{IE_{Q^f}} E_{Q^f} \xleftarrow{\eta_{Q^f}} Q(Y) \xrightarrow{\eta_Q} Q(X)$$

(a) $PO$ in $G || P$ (b) $PO$ in $G || P || O_E$ (c) $PO$ in $G || P || O_E$

Proof. The proof consists of iterated applications of Lemma 2 to the sequence of $m$-queries $Q_1, \ldots, Q_n$, which results in the sequence of substituted $m$-queries $Q'_1, \ldots, Q'_n$, together with respective pushout squares depicted in (39a). The pushout pasting lemma allows us to combine all these pushouts into a big pushout. We can apply the same principle not only horizontally but also vertically.
5 Basic Diagram Algebra: Operations without Dependencies

cally, which provides us with functoriality of $Q$.

\[
\begin{array}{c}
X = I_{Q_1} \xrightarrow{1_{E_Q}} E_{Q_1} \xrightarrow{1_{E_Q}} E_{Q_2} \cdots \xrightarrow{1_{E_Q}} E_{Q_n} = E_Q \\
Y = I_{Q_1} \xrightarrow{1_{E_Q}} E_{Q_1} \xrightarrow{1_{E_Q}} E_{Q_2} \cdots \xrightarrow{1_{E_Q}} E_{Q_n} = E_Q
\end{array}
\]

Corollary 10 (Category of complex queries). The collection of all $P$-sketch as objects (modulo their isomorphism) and $c$-queries (modulo $\cong$) as arrows form a category denoted by $CQ$:

\[CQ(X, X') = \{ Q \in cQQ(X) | \eta_Q: X \to X' \}\]

and identity $id_{CQ}(X) = 0_X: X \to X$.

5.4 Views and their Composition

Definition 23 (Views). Let $S$ be a $P$-sketch called the source, and $O$ an operation signature over $P$. An $O$-view of $S$ is a pair $v = (Q_v, \hat{v})$ with $Q_v$ a complex $O$-query over $S$ and $v: Q_v(S) \leftarrow V$ a sketch morphism called view (definition) mapping; the domain of the mapping is called the view schema. Thus, a view is a cospan $S \xrightarrow{\eta_v} Q_v(S) \xleftarrow{\hat{v}} V$, which we will write as $S \xrightarrow{\eta_v} Q_v(S) \xleftarrow{v} V$ to ease notation. Note that the left leg is monic while the right one is arbitrary. We will also write views as $v: V \to S$ or $v: V \to S$ to make the query explicit.

Theorem 3 (Category of Views (combinatorial Kleisli construction)). Views are associatively composable and thus make a category $S_O$ whose objects are sketches in $S = \mathbb{G}|||P$, and arrows are equivalence classes $O$-views between them.

This equivalence relation $\sim$ is defined as follows: Two views $S \xrightarrow{\eta_{v_1}} Q_{v_1}(S) \xleftarrow{v_1} V$ and $S \xrightarrow{\eta_{v_2}} Q_{v_2}(S) \xleftarrow{v_2} V$ are equivalent, $v_1 \sim v_2$, iff there exists an $\mathbb{G}|||P|||O_E$-isomorphism $i: Q_{v_1}(S) \to Q_{v_2}(S)$ such that $\eta_{v_1} \circ i = \eta_{v_2}$ and $v_1 = v_2$.

Proof. To define a category we have to define two operations (identities and composition) and show three properties to hold (left unitor, right unitor and associativity).

Composition:
Let $v_1: V \to S$ and $v_2: S \to T$ be two views with $V, S$ and $T$ being sketches in $\mathbb{G}|||P$. Their composition $v_1 \circ v_2: V \to T$ is defined as follows: Consider the diagram in (40): first, we employ the substitution Lemma 5 on p.44 to substitute $Q_{v_2}$ along $v_2$ into $Q_{v_1}$ resulting in the pushout square (a) and a new substituted...
complex query $Q_{v_1}^{c}$ with input $I_{Q_{v_1}^{c}} := Q_{v_1}(T)$. Evidently we can merge two adjacent complex queries into one complex query by concatenating their sequences of multiqueries. Thus, we get a complex query $Q_{v_2} \circ Q_{v_1}$ (or $Q_{v_1} \circ Q_{v_2}$) with arity monic

\[ IE_{Q_{v_1} \circ Q_{v_2}} = \eta_{v_2} \circ \eta_{v_1} : T \rightarrow Q_{v_1}(Q_{v_2}(T)). \]

Together with the composite $v_1 \circ v_2$ for the view mapping, we get the view $v_1 \circ v_2$.

\[ T \xrightarrow{v_2} Q_{v_2}(T) \xrightarrow{(a)} Q_{v_1}(Q_{v_2}(T)) \xleftarrow{\eta_{v_1}} S \xrightarrow{\eta_{v_2}} V \]

\[ Q_{v_1}(S) \xrightarrow{v_2^*} Q_{v_1}(Q_{v_2}(T)) \]

Identities:

For any sketch $S \in S$ we can consider a special complex query $Q_{1}^{S}$, whose sequence of multiqueries is empty and thus $I_{Q_{1}^{S}} = E_{Q_{1}^{S}}$ and $IE_{Q_{1}^{S}} = \eta_{Q_{1}^{S}} = id_{S}$.

Hence, we define the identity view $id_{S} : S \rightarrow S$ on $S$ as the span $S \xrightarrow{id} Q_{1}^{S}(S) = S \xleftarrow{id} S$.

Unitors:

\[ S \xrightarrow{id} \xrightarrow{id_{S}} S \]

\[ S \xrightarrow{v_1} \xrightarrow{\eta_{v_1}} Q_{v_1}(S) \]

\[ S \xleftarrow{\eta_{v_1}} \xleftarrow{id} Q_{v_1}(S) \]

Consider the diagrams in (41) depicting the compositions $v_1 \circ id_{S}$ and $id_{V} \circ v_1$ respectively, where $v_1 : V \rightarrow S$ is an arbitrary view and $id_{S}$ and $id_{V}$ are identity views over $S$ and $V$. Substituting $Q_{v_1}$ along $id_{S}$ (l) results in the same query and concatenating $Q_{1}^{S}$ (empty) with $Q_{v_1}$ results in $Q_{1}^{S}$ and since $v_1 \circ id = v_1$ it follows $v_1 \circ id_{S} = v_1$. On the other hand, substituting (r) an empty query $Q_{1}^{V}$ along a morphism changes the input $I_{Q_{1}^{S}}$ (which is also the output) and the sequence of multiqueries stays empty. Concatenating it with $Q_{v_1}$ results in $Q_{v_1}$ and again $id \circ v_1 = v_1$ and thus it follows $id_{V} \circ v_1 = v_1$.

Associativity:
Let \( v_1: V \rightarrow S \), \( v_2: S \rightarrow T \) and \( v_3: T \rightarrow U \) be three views, which are depicted in the upper half of (42).

We claim that \( (v_1 \bowtie v_2) \bowtie v_3 = v_1 \bowtie (v_2 \bowtie v_3) \), i.e. they both result in the same view: It does not matter if we first substitute \( Q_{v_1} \) along \( v_2 \) (a), concatenate with \( Q_{v_3} \), then substitute along \( v_3 \) (b) + (c) and finally concatenate with \( Q_{v_1} \), or if we first substitute \( Q_{v_2} \) along \( v_3 \) (a), concatenate with \( Q_{v_3} \), then substitute \( Q_{v_1} \) along \( v_2 \bowtie v_3 \) (a) + (c) to finally concatenate \( Q_{v_3} \bowtie Q_{v_2} \) with \( Q_{v_1} \). In both cases the result is a configuration of three uniquely determined PO squares (recall that our sketch categories are skeletal) and a complex query \( Q_3 \bowtie Q_2 \bowtie Q_1 = [Q_{v_3}^{(1)}, \ldots, Q_{v_3}^{(k)}, Q_{v_2}^{(1)}, \ldots, Q_{v_2}^{(l)}, Q_{v_1}^{(1)}, \ldots, Q_{v_1}^{(m)}] \) with unit \( \eta_{v_3} \bowtie \eta_{v_2} \bowtie \eta_{v_1} \) and view mapping \( v_1 \bowtie v_2 \bowtie v_3 \).

**Theorem 4 (Views form a SMC).** The category of \( \mathcal{O} \)-views \( \mathcal{S}_\mathcal{O} \) can be endowed with a symmetric monoidal structure \( (\bowtie, I) \) via coproducts and a (chosen) initial object in \( \mathcal{S} \).

**Proof.** To show has \( \mathcal{S}_\mathcal{O} \) has a symmetric monoidal structure we have to extend the proof in Thm. 3 with definitions of the monoidal product \( \bowtie \), the monoidal unit \( I \) and have to show that the monoidality laws hold.

**Monoidal Product:**

The monoidal product of two sketches \( S_1 \) and \( S_2 \) is simply their coproduct (note that \( \mathcal{S} \) is a topos). The monoidal product of two morphisms (= views)

\[
S_i, \xrightarrow{n_i} Q_i(S_i) \xleftarrow{v_i} V_i, i = 1, 2
\]
is specified by diagram (43) in \( S \):

\[
\begin{aligned}
\begin{tikzcd}
S' \arrow{r}{\eta_1} \arrow{d}{\iota_1} & S_1 \sqcup S_2 \arrow{r}{\iota_2} \arrow{d}{\eta_2} & S_2' \arrow{d}{\eta_2}
\end{tikzcd}
\end{aligned}
\]

\[
\begin{aligned}
\begin{tikzcd}
Q_1(S) \arrow{r}{\iota_1^*} & Q_1^*(S_1 \sqcup S_2) \arrow{d}{\eta_1^* + \eta_2^*} \arrow{r}{(Q_1^* + Q_2^*)(S_1 \sqcup S_2)} & Q_2(S_2) \arrow{d}{\eta_2} \arrow{d}{\kappa_2}
\end{tikzcd}
\end{aligned}
\]

\[
\begin{aligned}
\begin{tikzcd}
V_1 \sqcup V_2 \arrow{d}{v_1 \eta_1^*} \arrow{r}{v_2 \eta_2^*} & V_1 \sqcup V_2
\end{tikzcd}
\end{aligned}
\]

Here, \( S_1 \sqcup S_2 \) and \( V_1 \sqcup V_2 \) are coproducts with their respective injections, the diamonds \((l)\) and \((r)\) are substitution squares (Lem. 5), mapping \( \eta_1^* + \eta_2^* \) is a result of parallel composition (Lem. 4) and mapping \( u \) is given by the universal property of coproducts. Now the cospan \(((\eta_1^* + \eta_2^*), u)\) is defined to be the monoidal (parallel) composition of views:

\[
(v_1 \oplus v_2)_{Q_1 \oplus Q_2} : S_1 \oplus S_2 \leftarrow V_1 \oplus V_2
\]

We have to show that \( \oplus \) is a bifunctor a.k.a as the interchange law, i.e. for any four views \( v_1 : T \leftarrow W, v_2 : S \leftarrow V, v_3 : P \leftarrow T \) and \( v_4 : U \leftarrow S \) the following has to hold:

\[
(v_1 \oplus v_2) \Downarrow (v_3 \oplus v_4) = (v_1 \Downarrow v_3) \oplus (v_2 \Downarrow v_4)
\]  

(44)

The left hand side of this diagram is explicated in (45) below (the final result is highlighted in blue):
To show that (44) holds, we need to show that there exists an isomorphism between \( Q_{3|4} \) and \( Q_{1|3} \) that makes the cospan triangles commute. This isomorphism is provided to us by (37) in Lemma 9 which also makes the left hand side triangle commute and the universal coproduct property of \( V \sqcup W \) makes the right hand side triangle commute.

**Monoidal Unit:**

The monoidal unit is given by the empty sketch 0 ∈ \( \text{Ob}(S) \) (initial object in \( S \)).

**Monoidality:**

Note that \( S \) is a topos and thus it has a symmetric monoidal structure \((+, 0)\) given by coproduct and initial objects. Let now \( \alpha, \lambda, \rho \) and \( \sigma \) denote the associator, left-, right-unitor and braiding isomorphism families. Furthermore, there exists an identity-on-objects functor \( S \rightarrow \text{Ob}(S) \) that sends a sketch morphism \( f: A \rightarrow B \) to the trivial view \( B \sqcup 0_{B}(B) \sqcup A \). Functors preserve isomorphisms and thus the images of \( \alpha, \lambda, \rho \) and \( \sigma \) under this functor establish the coherence on \( S \).

### 5.5 Query Monad

**Background.** Complex queries are the diagrammatic equivalent of complex terms in the ordinary algebra, in which the term construction can be specified by the notion of a monad or, equivalently, a Kleisli triple, over some carrier category \( C \).

**Definition 24 ((Global) Kleisli triple).** A Kleisli triple over a category \( C \) is \( (Q, \eta, \_\circ) \) where \( Q: \text{Ob}(C) \rightarrow \text{Ob}(C) \) is an endofunction over \( \text{Ob}(C) \), \( \eta \) a family
of \( \mathbb{C} \)-arrows \( \eta_S : X \to Q(X) \), \( X \in \text{Ob}(\mathbb{C}) \), and \( \cdot^* \) is an operation that send any \( v : Q(S) \to V \) in \( \mathbb{C} \) to \( v^* : Q(S) \to Q(V) \) illustrated by the diagram below:

\[
\begin{array}{c}
S \xrightarrow{\eta_S} V \xleftarrow{\eta_V} W \\
Q(S) \xrightarrow{- \cdot v^*} Q(V) \xleftarrow{- \cdot w^*} Q(W)
\end{array}
\] (47)

such that the following three equation hold: (a) \( \eta_X^* = \text{id}_X \), (b) \( \eta_Y^* \cdot v^* = v \) and (c) \( (w \cdot v^*)^* = w^* \cdot v^* \).

Such a triple is called \textit{mono} if all \( \eta_X \) are monic.

A triple is \textit{finitary} if \( Q \) commutes with filtered colimits: \( Q(\text{colim} \{ X_i \mid i \in \mathbb{I} \}) = \text{colim} \{ Q(X_i) \mid i \in \mathbb{I} \} \) for some functor \( X : \mathbb{I} \to \mathbb{C} \) from a filtered category \( \mathbb{I} \).

Note that Kleisli arrows above are defined \textit{globally} in the sense that object \( Q(S) \) is generated by applying \textit{all} complex terms/queries and different views \( v, w \) use the same construction on the target. In contrast, our category of views \( S \) in Thm. 3 is defined \textit{locally}: each view carries its own query necessary to define the view mapping as illustrated below.

\[
\begin{array}{c}
S \xrightarrow{v} V \xleftarrow{w} W \\
Q_v(S) \xrightarrow{(\ast)} Q_w(V) \\
\eta_w \xrightarrow{\eta_w^*} \eta_v \rightarrow Q_{w\cdot v}(S)
\end{array}
\] (48)

where square \( (\ast) \) is provided by the query substitution (based on Lemma 5) as explained above.

Given a sketch \( X \in S \), let \( cQQ_n(X) \) be the set of all complex queries \( Q \) over \( X \) with \( I_Q = X \) and the length equal or less than \( n \). We thus have a chain

\[
cQQ_1(X) \subset cQQ_2(X) \subset ...
\]

of wide spans. Taking wide PO of each of them gives us a commutative diagram of monics in \( \mathbb{G}/|\mathcal{P}|\mathcal{O}_E \); \( \eta_n \cdot t_n = \eta_{n+1} \), \( n = 1, 2, ... \)

\[
\begin{array}{c}
X \xrightarrow{\eta_1} Q_1(X) \xrightarrow{\eta_2} Q_2(X) \xrightarrow{\eta_3} Q_3(X) \rightarrow ... \\
\end{array}
\] (49)

whose colimit is a \( \mathbb{G}/|\mathcal{P}|\mathcal{O}_E \)-monic denoted by \( \eta_{X}^X : X \to Q_X(X) \).

Now Kleisli composition is specified by the following diagram in Fig. 12, where the middle diamond is a query substitution PO provided by Lemma 5.

The Kleisli star operation requires the existence of arrow \( v^*_Q \), and such would be provided by the universal property of \( Q(X) \) if for each query \( Q \) over \( V \), we would have the diagonal arrow \( d_{Q,v} \) as shown in the diagram, which in turn would be provided by the \textit{monadic multiplication} arrow \( \mu_{Q,v} \), so that we could define \( d_{Q,v} := v^*_Q \cdot \mu_{Q,v} \). The existence of such arrows \( \mu \) is a typical feature of
5 Basic Diagram Algebra: Operations without Dependencies

the term formation procedure ("a term over a term is a term"), but if the image of \( v \) spans the entire \( Q_x(S) \), this construction does not work. Suppose, e.g., that \( V = \{ x_1, ..., x_n, ... \} \) is countable and \( v(x_n) \in cQQ_n(S) \) for all \( n \). Then, if a query \( Q \) over \( V \) uses all "variables" \( x_i \), then its substitution \( Q^f \) along \( v \) will span the entire \( Q_x(S) \) and disable the construction. To make it working, we need to refine the query substitution mechanism and introduce some restrictions on the number of variables (i.e., the size of the sketch) involved into queries.

Effective Substitutions. Recall that presheaf toposes have image factorization, and hence we can factorize the binding square for atomic queries into two POs as shown below, where \( I_q^\text{eff} \) denotes the image of \( b_q \) in sketch \( I_q \).

\[
\begin{array}{c}
\text{l} \quad \text{I}_q \quad \text{I}_q^\text{eff} \quad \text{I}_q^\text{eff} \quad \text{l} \\
\downarrow \quad \downarrow \quad \downarrow \quad \downarrow \\
\text{E}_q \quad \text{E}_q^\text{eff} \quad \text{E}_q^\text{eff} \quad \text{E}_q^\text{eff} \quad \text{E}_q \\
\end{array}
\]

This factorization can be taken into account for m-query and c-query substitution, and results in the following.

Lemma 6 (Effective Substitutions) For any \( \mathcal{P} \)-sketch \( X \) and query \( Q \) over \( X \), we have a composition of three POs as shown in diagram (a) below, where we write \( \eta^\text{eff}_Q \) instead of an accurate but monstrous \( \eta^\text{eff}_Q \). Diagram (b) is the \( \text{G}||\mathcal{P} \)-projection of (a).

\[
\begin{array}{c}
\text{l} \quad \text{I}_q \quad \text{I}_q^\text{eff} \quad \text{I}_q^\text{eff} \quad \text{l} \\
\downarrow \quad \downarrow \quad \downarrow \quad \downarrow \\
\text{E}_q \quad \text{E}_q^\text{eff} \quad \text{E}_q^\text{eff} \quad \text{E}_q^\text{eff} \quad \text{E}_q \\
\end{array}
\]
A triple is equivalent to a finitary monad satisfying the McSCert report 2021-34.

We thus come to the following result.

\[ \eta \]

is straightforward; property (c) is our local Kleisli composition defined above.

A finitary diagrammatic query-building mechanism (Def. 25 above) gives rise to a finitary \( \eta \)-mono (global) Kleisli triple \( (Q_\mathcal{X}, \eta_\mathcal{X}, \_\mathcal{X}) \) whose components were defined above. As is well-known, such a triple is equivalent to a finitary monad satisfying the \( \eta \)-mono requirement.

Now we introduce necessary size restrictions. Rather than doing this in a general setting with cardinal arithmetics, we will do it for the simple finitary case (but the pattern is generalizable for any infinite cardinal).

**Definition 25 (Finitary Queries).** An operation signature \( \mathcal{O} \) is called finitary if for all \( \phi \in \mathcal{O} \), sketch \( E_\phi \) (and hence \( I_\phi \)) is finite. A multiquery \( Q \) is finitary if \( \mathcal{O} \) is finitary and set \( \text{Index}(Q) \) is finite. A complex query \( Q \) is finitary if all \( m \)-queries in the sequence \( (Q_1, ..., Q_N) \) are finitary.

**Corollary 11 (Finitarity and Substitutions).** If query \( Q \) over \( X \) is finitary, then sketch \( Q(X_{eff}) \) is finite. Moreover, for any mapping \( f: x \to Y \), the sketch \( Qf^{eff}_{\mathcal{Q}}(Y_{eff}) \) (and hence \( E^{eff}_{\mathcal{Q}} \)) is finite too.

Now we can implement the proof idea outlined above. For any global view \( v: Q_\mathcal{X}(S) \to V \) and a finitary query \( Q \) over \( V \), we apply Lemma 6 on effective substitution and build the diagram in Fig. 13 (ignore the dotted arrows for a moment), where we use a new lighter notation for the two intermediate sketches. As sketch \( Q'_{\mathcal{X}} \) is finite, there is a number \( N \) (depending on \( Q, f \)) such that \( V'_{\mathcal{X},v} \subset Q_N(S) \), and hence sketch \( Q'_{\mathcal{X},v} \) can be mapped to \( Q_{N+1}(S) \subset Q_\mathcal{X}(S) \) (note arrow \( \mu_{Q,v} \)). Composition of \( \mu_{Q,v} \) with \( (v^{eff})_Q \) gives us \( d_{Q,v} \) for each query \( Q \) over \( V \), and thus a cocone with vertex at \( Q_\mathcal{X}(S) \). However, \( Q_\mathcal{X}(V) \) is a universal (minimal) cocone over \( \mathcal{X} \) same base, and hence we have a unique arrow \( d_{Q,v} \). This arrow can be taken as the result the global Kleisli triple operation \( \_\mathcal{X} \) (denoted by \( \_\mathcal{X} \) in our notation). Checking monoidal properties \((a,b)\) of Def. 24 is straightforward; property (c) is our local Kleisli composition defined above. We thus come to the following result.
6 Advanced Diagram Algebra: Dependencies between Operations

In the previous section we investigated diagram algebra for the discrete case, i.e. \( \mathcal{O} \) being a set. Now we consider signatures of operations with dependencies, which will provide us with reasoning capabilities. We will begin with a motivating example.

**Example 8 (Universal Pullback Property).** To specify commutativity of the pb-square, we need to a) extend the signature \( \mathcal{P}_{\text{monic}} \) with a new predicate symbol \([;] \) whose input arity \( I_{[;]} \) are two consecutive arrows and its scope arity is the triangle \( E_{[:]} \), and b) add to the sketch \( E_{[\text{pb}]} \) a new arrow (the diagonal of the square) plus two \([;] \)-constraints declaring the two triangles dividing the square to be commutative. Thus, \( E_{[\text{pb}]} \) becomes a \{[monic], [;]\}-sketch. Now, if we want to specify the universal property of pullbacks, we add to the signature a new operation symbol \([\text{upb}] \), whose input arity is a sketch in the signature \( \mathcal{P}_{\text{pb}} = \{[\text{monic}], [\text{pb}], [;] \} \) (where \([\text{pb}] \) is a predicate symbol with arity \( E_{[\text{pb}]} \)). Fig. 14b depicts the arity of this \([\text{upb}] \). Black-colored elements are part of \( I_{[\text{upb}]} \) and blue-colored elements are added by \( E_{[\text{upb}]} \). The input sketch comprises two squares together with respective diagonals and commutativity conditions expressed via \([;] \)-triangles, additionally one of the squares has a \([\text{pb}] \)-predicate. The scope sketch adds a new arrow \( u \) together with two commutative triangle predicates \([;] \).

A formal specification of the above is to found below
(a) Categorical Notation

(b) Arity of [upb]

(c) Workflow

Fig. 14: Example: Universal Pullback Property
6 Advanced Diagram Algebra: Dependencies between Operations

\[
\begin{array}{c}
I_{\text{label}(q)} \xrightarrow{IE_{\text{label}(q)}} E_{\text{label}(q)} \\
\downarrow \quad \downarrow \\
I_Q \xrightarrow{IE^q} E_Q \\
\downarrow \quad \downarrow \\
f \xrightarrow{IE^q_f} E_{Q^f} \\
X' \xrightarrow{IE_{Q^f}} E_{Q^f}
\end{array}
\]

(31)
Figure 15 presents a formalized description of the above. We have a predicate signature $O_{\text{upb}}$ (see diagram (a)) consisting of a predicate symbol [monic] and above it an operation signature $O$ consisting of three operation symbols {[:, [pb], [upb]}. There are two types of dependencies between symbols. ** Preconditions (shown in orange) specify conditions to make an operation applicable, and any such dependency arrow is assigned with a mapping of the scope sketch of the dependency’s target into the input sketch of the dependency’s source. ** Postconditions (shown in blue) specify conditions that the result of operation must always satisfy, and any such dependency arrow is assigned with a mapping between the scopes of the related symbols.

![Diagram](image.png)

(a) Signature $O_{\text{upb}}$

(b) Arity Functors

** Fig. 15: Category $O_{\text{upb}}$

### 6.1 Functorial Operation Signatures (with Dependencies)

We will need the notion of diagonal arrow category described in Def. 31, 32 on p. 65.

** Definition 26 (Operation Signature with Dependencies).** An operation signature $O$ is given by two contravariant functors $\alpha_{\text{IE}}: O_{\text{prec}}^{\text{op}} \to [\vdash, S]_{\text{diag}}$ and $\alpha_{\text{EE}}: O_{\text{post}} \to S$ where $O_{\text{prec}}$ is a subcategory of $O_{\text{post}}$ sharing the same set of objects and the diagram (52) in $\textbf{Cat}$ commutes.

$$O_{\text{prec}}^{\text{op}} \xrightarrow{\alpha_{\text{IE}}} [\vdash, S]_{\text{diag}} \xleftarrow{\alpha_{\text{EE}}} \text{cod} \xrightarrow{\text{cod}} S$$

(52)
The diagram category $\mathcal{D}_{\text{diag}}$ is a subcategory of $\mathcal{D}$ with chosen diagonals, thus every morphism $(\Delta_I, \Delta_E): \delta \Rightarrow \delta'$ in $\mathcal{D}_{\text{diag}}$ is identified by a morphism $d: \delta(E) \to \delta'(I_k)$ in $\mathcal{S}$. Furthermore, we call arrows in $\text{Arr}(\mathcal{O}_{\text{pre}})$ preconditions and those in $\text{Arr}(\mathcal{O}_{\text{post}})$ postconditions. However, amongst those, only arrows in $\text{Arr}(\mathcal{O}_{\text{post}})^{-}\text{Arr}(\mathcal{O}_{\text{pre}})$ are really interesting postconditions, and later by postconditions we will only mean the latter.

**Diagrammatic Algebras and Queries (dependencies)**

\[ \text{(53)} \]

**Definition 27 (O-algebra).** Given an operation signature $\mathcal{O}$, an $\mathcal{O}$-algebra $A$ is given by a sketch $|A|$ and two mappings $\llbracket - \rrbracket_A^{\text{pre}}$ and $\llbracket - \rrbracket_A^{\text{post}}$, see (53), where $\text{LDiagr}(A)_{\text{pre}}$ is given by the pullback of $\downarrow_{\text{IE}}$ and $\alpha_{IE}$ and $\text{LDiagr}(A)_{\text{post}}$ by the pullback of $\downarrow_{\text{IE}}$ and $\alpha_{IE}$.

**Definition 28 (O-multiquery).** Given an operation signature $\mathcal{O}$, an $\mathcal{O}$-multiquery is given by two spans $(\text{diag}_{\text{pre}}, \text{label}_{\text{pre}})$ and $(\text{diag}_{\text{post}}, \text{label}_{\text{post}})$ where the apex of the former $\text{Index}(Q)_{\text{pre}}$ is a subcategory of $\text{Index}(Q)_{\text{post}}$.

### 6.2 Hierarchical Operation Signatures and Queries

In this section, the previous categorical/functorial definitions are restated in the discrete but hierarchical way.

**Definition 29 (Hierarchical operational signatures).** Let $\mathcal{G}$ be a presheaf topos of $\mathcal{G}$-graphs as before. A predicate dh-signature of height $N$ is given by the inductively defined data specified by the commutative diagram (54).

\[ \text{(54)} \]

The upper row is provided by a discrete hierarchical category $\mathcal{O}^*$ of predicate symbols of height $N$, i.e., a finite chain of non-empty sets shown in the top row.
of the diagram. Each of these sets $O^*_i$ is equipped with an arity function $\alpha^*_i$ into the corresponding class of sketch monics

$$S_{i-1} \overset{\text{def}}{=} S_{i-2} \parallel O^*_{s_{i-1}}, \ i < N$$

as shown in the bottom row of the diagram (see Def. 18 on p. 34 that explains the notation). The sequence starts from $i = 0$ for which $S_{-1} = G \parallel P$, i.e., sketches without operation symbols, which allows us to define mapping $\alpha^*_0$, which in turn defines sketch category $S_0$ and so on. Recall that diagonal dotted arrows labelled $\text{:sig}$ are pairs rather than mappings.

The above defines a dh-signature $O^*$. Sketches over $O^*$ and their morphisms are defined as objects and arrows of the category $S_{N-1} \overset{\text{def}}{=} S_{N-2} \parallel O^*_{s_{N-1}}$. We will denote this category by $G \parallel O^*$: strictly speaking, we should write $G \parallel O^*$ as it’s another construct of building sketches, but we will usually omit the superscript near $\parallel$ if the right term unambiguously shows the dh-context of $\parallel$.

TO BE COMPLETED
7 Conclusion and Future work

To be completed.
References


Appendix

A Class Diagrams and their Mappings

A.1 Domain Modelling with Class Diagrams.

Figure 16 (on page 62) presents a toy example of domain modelling with UML class diagrams [29]. The source model $S$ is specified by a UML class diagram describing a part of a university information structure in an almost self-explanatory
way. Boxes refer to classes of objects Person, Course etc. to be instantiated by object identifiers, and show their attributes (name, workload, etc.) instantiable by functions, e.g., for an object \( o \in \text{Person} \) with \( \text{name}(o) = \text{Bill Clinton} \) (where \( [x] \) denotes a semantic interpretation of an element \( x \) of the diagram). Arrows denote associations to be instantiated by mappings (relations or functions), and expressions \( [m.n](m,n \in \{0,1,2,\ldots;\ast\}) \) (shown in red) near associations ends are called multiplicities and constrain the type of relation/function, e.g. multiplicity \([3..5]\) at the source end of association student says that each student may only have from 3 to 5 subscriptions, i.e., for a valid instantiation \([\cdot]\), for any object \( s \in \text{Student} \) the following condition is to hold:

\[
\text{card}\left(\{b \in \text{Subscription} | \text{student}(b) = s\}\right) \in [3..5],
\]

and if this condition is violated, instantiation \([\cdot]\) is considered invalid (and some alarm is to be fired).

![Fig. 16: Sample UML diagram](image)

Constraints often appear as implementations of a set of requirements (typically conflicting), e.g., the Student Health Office may require the multiplicity above to be \([2..5]\) while the Graduate Office requires it to be \([3..\ast]\). Keeping the backward traceability to requirements is important and often crucial for a proper domain modelling. By default (and thus omitted in Fig. 16), every association arrow has the target multiplicity \([1..1]\) (i.e. exactly one) and the source multiplicity \([0..\ast]\) (i.e. arbitrary), which makes the relation a function.

Furthermore, there are different arrow-types, e.g. the arrows between Employee and UniversityRole, and between Student and UniversityRole, show the inheritance (subset) relationship between classes. It means that both Employee and Student inherit the person association from UniversityRole. Multiplicities constraints and inheritance are typical examples of builtin-constraints in UML. However, there may be more complex domain-specific rules that cannot be expressed with builtin constraints and require special attached constraints commonly formulated in the Object Constraint Language (OCL) [31]. For example, requirement Req 3 that student teaching assistants are not allowed to earn more than 15,000.00 a year due to tax regulations would be formalized as shown in Listing 1 and indicated in Fig. 16 with a note-symbol attached to class Employee. We can express this constraint in an element-free way by taking suitable pullbacks along inheritance.
A Class Diagrams and their Mappings

monics and the monic \{15000\} \to \text{Real}, but it would be a bulky and opaque construction for a practical engineer. Instead, in our generalized sketch framework, we consider such an OCL constraint as a formal constraint declaration \(C\) whose scope \(\text{diagr}(C)\) consists of the inheritance triangle (i.e., the corresponding cospan) and attribute \text{salary} (considered as an arrow from \text{Employee} to \text{Real}). Semantics of the constraints is the Boolean-valued procedure specified in the Listing, which classifies any configuration of sets and mappings over the scope as either valid or invalid.

Listing 1: OCL constraint

```ocl
context Employee inv TA_Payment:
  Student.allInstances()->exists(s | s.person = self.person)
  implies self.salary <= 15000.00
```

A.2 View definition example.

An application working with the university database may only need a part of information stored in the database (due, say, to security concerns), e.g., it may only need data about students and courses they take. Such data are specified by schema \(V\) specified in the right part of the figure, which is called a view of \(S\). To make the view definition formal, we need to map schema \(V\) to schema \(S\), but there is an issue: association \text{takes} is not present in \(S\). The issue is resolved by noticing that the derived association \(\text{student}^{-1}\).\text{course} produced by, first, inverting \text{student}, and then composing it with association \text{course}, is actually what view \(V\) means by \text{take}. Multiplicities of the new association are also derivable. In other words, by applying operations/queries \(q_1\) (inversion) and \(q_2\) (composition), we derive new elements of model \(S\) shown in blue (to suggest the mechanical nature of computation) and dashed (for the black-white printing); note that queries \(q_1, q_2\) are diagram operations of obvious diagrammatic arities. They can be defined in OCL as shown in Listing 2 (where \(\text{student}^{-1}\) is called \text{student_inverse} and \(\text{student}^{-1}\.\text{course}\) is called \text{student_to_course} due to a limited character palette).

Listing 2: Derived elements in OCL

```ocl
context Student :: student_inverse : Subscription[2..*]
derive Subscription.allInstances() \to select(s | s.student = self)

c
context Student :: student_to_course : Course[2..*]
derive Subscription.allInstances() \to select(s | s.student = self)
  .collect(s\text{.course})
```

Now we can build the view definition mapping by mapping arrow \text{takes} in \(V\) to the derived arrow \(\text{student}^{-1}\.\text{course}\) in \(S\). Formally, we express this as a mapping \(v: Q_v(S) \to V\) where \(Q_v = \{q_1, q_2\}\) and \(Q_v(S)\) is the full (black, red and blue) model/sketch shown in the figure, whereas the source model is the (black and red) subsketch \(S\) not including the derived arrows and derived constraints.

Importantly, the view definition mapping \(v\) is compatible with constraints:

\[(\text{context :: student}^{-1}\.\text{course})\mid [3..5] |\=[2..5]\]

in the sense that any instantiation \([\square]\_S\) satisfying \([3..5]\) will also satisfy \([2..5]\) and hence its view \([\square]\_V\) also satisfies \([2..5]\) as required: compatibility of a view mapping with constraints ensures the correctness of views. Thus, an important requirement to a view definition mapping \(v\) is to be a sketch morphism, so the category hosting our artifacts is to be a category of sketches rather than graphs.
B  Diagram Categories

We will recall a few basic definitions and results and provide several new auxiliary constructs needed in the paper.

A diagram in a category $\mathcal{C}$ is a functor $\delta: \mathcal{D} \to \mathcal{C}$ whose domain is a category freely generated by a graph $D$ called the shape of the diagram (and $\mathcal{D}$ is also called the shape). Sometimes we will make $D$ a sketch by adding to it corresponding declarations, e.g., a valid diagram of shape $\begin{array}{c} \xymatrix{ a \ar[r] & b } \end{array}$ must map two horizontal arrows into monics, and the entire square into a pushout square in $\mathcal{C}$. A precise formalization of this construct in general is exactly the subject of the paper, but in the metatheory we will only use simple sketches like above that can well be explained like above.

A diagram morphism is a natural transformation between functors, $\Delta: \delta \Rightarrow \delta'$, i.e., a family of arrows $\Delta_x: x^\delta \to x'^\delta$ indexed by $x \in \text{Ob}(\mathcal{D})$ such that $a^\delta \cdot \Delta_y = \Delta_x \cdot a'^\delta$ for any arrow $a: x \to y$ in $\mathcal{D}$. This gives us the category $[\mathcal{D}, \mathcal{C}]$ of all diagrams in $\mathcal{C}$ of shape $\mathcal{D}$.

In subsection Sect. B.1 we will consider categories of diagrams whose shape is a single arrow.

B.1 Arrow categories.

**Definition 30 (Arrow Categories).** Any category $\mathcal{C}$ gives rise to an arrow category $[\rightarrow, \mathcal{C}]$ whose objects are $\mathcal{C}$-arrows and morphisms are commutative squares: given two $\mathcal{C}$-arrows $a, b$, their morphism is a pair of $\mathcal{C}$-arrows, $f: \bigcirc a \to \bigcirc b$ and $f': a\bigcirc \to b\bigcirc$, such that the square commutes: $a \cdot f' = f \cdot b$. We call such a pair an sq-arrow and write $p_{f,f'}: a \Rightarrow b$.

Any object $X \in \mathcal{C}$ gives rise to a dom-pointed arrow category $[x \rightarrow, \mathcal{C}] \subseteq [\rightarrow, \mathcal{C}]$ whose objects are arrows from $X$ and the corresponding component of all sq-arrows is $\text{id}_X$. Similarly, we have cod-pointed arrow category $[\rightarrow X, \mathcal{C}] \subseteq [\rightarrow, \mathcal{C}]$. These categories usually denoted by $\mathcal{C}\backslash X$ and $\mathcal{C}/X$ resp.

The following facts can be proven easily:

**Fact. 1** There are two obvious functors $\text{dom}: [\rightarrow, \mathcal{C}] \to \mathcal{C}$ and $\text{cod}: [\rightarrow, \mathcal{C}] \to \mathcal{C}$, for which the following holds.

i) For any $\mathcal{C}$, functor $\text{dom}$ is a split fibration (by precomposition).

ii) The restriction $\text{dom}_X: \mathcal{C}/X \to \mathcal{C}$ (for a given object $X$) is a discrete fibration.

iii) For any $\mathcal{C}$, functor $\text{cod}$ is a split opfibration (by postcomposition).

iv) The restriction $\text{cod}_X: \mathcal{C}\backslash X \to \mathcal{C}$ is a discrete opfibration,

When furthermore $\mathcal{C}$ has all pushouts (pullbacks) we can build more.

**Fact. 2** i) If $\mathcal{C}$ has all pushouts, the functor $\text{dom}$ is an opfibration as well where lifts are constructed via pushouts.

ii) If $\mathcal{C}$ has all pullbacks, the functor $\text{cod}$ is a fibration as well where lifts are constructed via pullbacks.

In Sect. 6.1 we will heavily use the following construct.
**Definition 31 (Diagonal arrow categories).** In an arrow category $[\rightarrow, C]$, a sq-arrow $(f, f')$: $a \Rightarrow b$ is called diagonal if there is a “diagonal” arrow $d$: $a \circ b$ such that $f = a \circ d$, $f' = d \circ b$; then we write $(f, f') = d^2$.

The associativity of composition ensures the following result.

**Lemma 7** Composition of two diagonal arrows is again diagonal: if $(f_1, f'_1) = d_1^2$: $a \Rightarrow b$ and $(f_2, f'_2) = d_2^2$: $b \Rightarrow c$, then $(f_1, f'_1) \circ (f_2, f'_2) = (d_1 \circ b \circ d_2)^2$.

**Definition 32 (Diagonal subcategories).** Given a category $C$, category $[\rightarrow, C]_{\text{diag}}$ has the same objects as $[\rightarrow, C]$, diagonal sq-arrows as arrows between different objects (i.e., different $C$-arrows), and the only looping sq-arrows $(f, f')$: $a \Rightarrow a$ being (non-diagonal) identity sq-arrows $f = \text{id}(\circ a)$, $f' = \text{id}(\circ a)$.

Category $[\rightarrow, C]_{\text{diag}}$ is a full subcategory of $[\rightarrow, C]_{\text{diag}}$, whose objects are monic arrows in $C$, and we have the following obvious commutative diagrams:

![Diagram](55)

with labels $\text{[wide]}$ and $\text{[full]}$ referring to wide and full embedding resp. Note that functor $\text{cod}'$ is faithful: if two parallel diagonals are different, then their compositions with the target monic arrow are different too. To ease notation, below we will denote all four $\text{dom}$-functors and all four $\text{cod}$-functors by name $\text{dom}$ and $\text{cod}$, resp., without primes.

---

## C Presheaf Toposes

In Computer science and Software Engineering context, *presheaf toposes* or *presheaves* have also been referred to as “graph-like structures”, e.g. [24].

There is an important difference between sets and presheaves, the latter cannot usually not be represented as a coproduct (sum) of its elements.

**Definition 33 (Presheaf).** Let $S$ be a small category. A functor $G: S \rightarrow \text{Set}$ is called an $S$-presheaf or just a presheaf if $S$ is clear from the context or is not important, and the category $\text{Set}^S$ of all such presheaves and their natural transformations is called a *presheaf topos*. $^9$

A presheaf $G: S \rightarrow \text{Set}$ is called finite if all sets $G(v)$, $v \in \text{Ob}(S)$ are finite.

A presheaf topos is called finitary if the schema category $S$ is finite. (Makaki says that in ref [3] in his paper 1, such toposes are called finite.

**Fact. 3 (Presheaf Topos)** *Any category of presheaves is a topos and thus has*

- all limits and colimits,

---

$^9$ Usually presheaves are defined as contravariant functors $G: S^{\text{op}} \rightarrow \text{Set}$ due to their relation and importance for the Yoneda Lemma. For our applications throughout this paper, it is more convenient to work with the covariant definition.
References

- exponents, i.e., is cartesian closed,
- epi-mono factorizations,
- a subobject classifier.

If \( S \) is \( \text{Graph} \), then 'graphs' are nothing but ordinary directed multigraphs, and if \( S \) is \( \text{rGraph} \), then 'graphs' are reflexive graphs. If \( S \) is \( \text{PNet} \), 'graphs' are Petri nets (bipartite graphs of places and transitions).

Several fixed schema categories we will use are shown in Fig. 17. Objects and arrows in these categories are fixed and have fixed names.

Presheaves in many ways behave as sets and we will make use of some set-theoretic concepts throughout the paper. As we have already said, we can speak of elements \( x \in G \) meaning that there exists an object \( I \in \mathbb{I} \) such that \( x \in G(I) \). Categorically, such a (global) element is modelled by an arrow \( x: 1 \rightarrow G \).

**Inclusions** Let \( G \) be an object in \( \mathbb{G} \). Two monics \( x: X \rightarrow G \) and \( y: Y \rightarrow G \) are equivalent, \( x \cong_G y \), if there exists an isomorphism \( i: X \rightarrow Y \) such that the triangle commutes, \( i \circ y = x \). An \( \cong_G \)-equivalence class is called an (abstract) subobject of \( G \). Thus, any monic \( x: X \rightarrow G \) gives rise to a subobject \( [x]: X \rightarrow G \). Among the members of the class \( [x]: X \rightarrow G \), there is a monomorphism \( \subseteq: H \rightarrow G \) whose domain \( H \) consists only of elements which are also elements in \( G \) — we call it inclusion and denote by \( H \hookrightarrow G \). Thus, for any object \( I \in \mathbb{G} \), we have subsetting \( H(I) \subseteq G(I) \).

**Images** The third an final concept we are borrowing from set theory are images. Categorically, the are given by epi-mono factorizations. Let \( f: H \rightarrow G \) be an arbitrary arrow in \( \mathbb{G} \), because \( \mathbb{G} \) has epi-mono factorizations \( f \) is equal to the composite \( e \circ m \) where \( e \) is an epimorphism and \( m \) is a monomorphism. The codomain \( \text{Img}(f) \) object of \( e \) (domain of \( m \)) is called the image and it is unique up to isomorphism, thus we can choose \( m \) to be an inclusion morphism. \[ \text{ZD} \] but see my remark above! \[ \text{ZD} \] The situation is depicted in the following diagrams.

![Fig. 17: Fixed shape categories](image-url)
Subobject lattices From the above it follows that for every object $G \in \mathcal{G}$, there exists a subobject lattice denoted $\text{Sub}(G)$. It is given by the subcategory $\text{Sub}(G) \subseteq \mathcal{G}/G$ of the slice category whose objects are abstract subobjects, meets are given by pullbacks in $\mathcal{G}$, and joins are given by $\text{Img}(\text{pr}_\mathcal{H}, \text{pr}_\mathcal{K})$, the image of the universal coproduct arrow for $H \rightarrowtail H + K \leftarrowtail K$ being the coproduct of the abstract subjects $H \in \text{Sub}(G)$ and $K \in \text{Sub}(G)$. Furthermore, for a morphism $f: H \rightarrowtail G \in \mathcal{G}$ we denote the pullback-functor (change of base) on the subject lattices of $G$ and $H$ by $f^{-1}: \text{Sub}(G) \rightarrow \text{Sub}(H)$. Note, that this functor has both a left (post-composition) adjoint $\exists f: \text{Sub}(H) \rightarrow \text{Sub}(G)$ and a right (bundling) adjoint $\forall f: \text{Sub}(H) \rightarrow \text{Sub}(G)$. Shorter: Any topos is a regular category. Hence, we have adjunctions $\exists f \dashv f^{-1} \dashv \forall f$ (see Freyd-Scedrov). For presheaf toposes, these adjunctions are given component-wise.

Adhesiveness Thus, it is important to recall that every presheaf topos $\text{Set}^\mathcal{B}$ is adhesive and adhesiveness is closed under the “operations” product-, slice, coslice- and functor-category [22].

In every adhesive category the following facts hold

Fact. 4 (Adhesive Category Properties [22]) – Monomorphisms are stable under pushouts.

– Pushouts along monomorphisms are pullbacks.
– Pushout complements of monos are unique up to isomorphism.
– Pushout can be decomposed along pullbacks.

These properties are required to establish the major theorems in DPO rewriting regarding Concurrency, Parallelism, Embedding and Critical Pairs of productions, for details see [15]. Critical Pairs are important for analysis of confluence. In general, confluence for a DPO rewriting system known to be terminating is undecidable [28]. However, when considering a variation of DPO with interfaces [19] confluence is decidable [4]. This fact is basis for theorem proving for SMTs described in [3].

McSCert report 2021-34 67 May 11, 2021