OLOGS: A CATEGORICAL FRAMEWORK FOR KNOWLEDGE REPRESENTATION

DAVID I. SPIVAK

Abstract. In this paper we introduce the olog, or ontology log, a category-theoretic model for knowledge representation (KR). Grounded in formal mathematics, ologs can be rigorously formulated and cross-compared in ways that other KR models (such as semantic networks) cannot. An olog is similar to a relational database schema; in fact an olog can serve as a data repository if desired. Unlike database schemas which are generally difficult to create or modify, ologs are designed to be user-friendly enough that authoring or re-configuring an olog is a matter of course rather than a difficult chore. It is hoped that learning to author ologs is much simpler than learning a database definition language, despite their similarity. We describe ologs carefully and illustrate with many examples. As an application we show that any primitive recursive function can be described by an olog. We also show that ologs can be connected together into a larger network using functors. Sheaf semantics could then be employed to study local vs. global world-views. We finish by providing several different avenues for future research.

Contents

1. Introduction

Scientists have a pressing need to organize their experiments, their data, their results, and their conclusions into a framework such that this work is reusable, transferable, and comparable with the work of other scientists. In this paper, I will discuss the "ontology log" or *olog* as a possibility for such a framework. Ontology is the study of what something *is*, i.e the nature of a given subject, and ologs are designed to record the results of such a study. The structure of ologs is based on a branch of mathematics called category theory. An olog is roughly a category that models a given real-world situation.

This project was supported by ONR grant: N000141010841 and a generous contribution by Clark Barwick, Jacob Lurie, and the mathematics department at MIT.

The main advantages of authoring an olog rather than writing a prose description of a subject are that

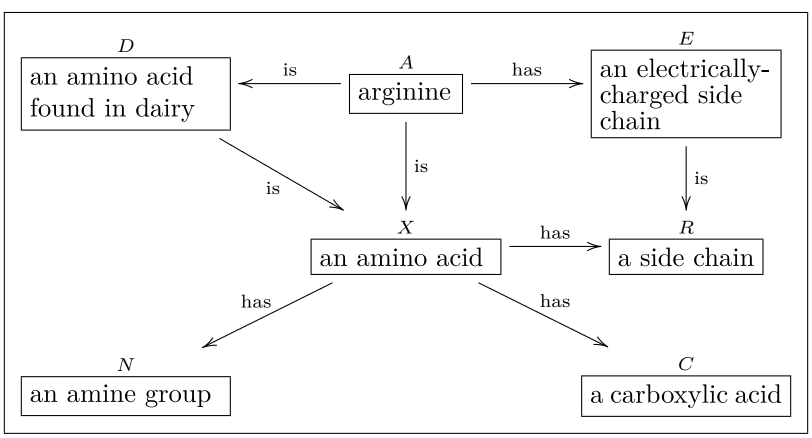
- an olog gives a precise formulation of a conceptual worldview,
- an olog can be formulaically converted into a database schema,
- an olog can be extended as new information is obtained,
- an olog written by one author can be easily and precisely referenced by others,
- an olog can be input into a computer and "meaningfully stored", and
- different ologs can be compared by functors, which generate automatic translation systems.

The main disadvantage to using ologs over prose, aside from taking more space on the page, is that writing a good olog demands a clarity of thought that ordinary writing or conversation can more easily elide. However, the contemplation required to write a good olog about a subject may have unexpected benefits as well.

A category is a mathematical structure that appears much like a directed graph: it consists of objects (often drawn as nodes or dots, but here drawn as boxes) and arrows between them. The feature of categories that distinguishes them from graphs is the ability to declare an equivalence relation on the set of paths. A functor is a mapping from one category to another that preserves the structure (i.e. the nodes, the arrows, and the equivalences). If one views a category as a kind of language (as we shall in this paper) then a functor would act as a kind of translating dictionary between languages. There are many good references on category theory, including [LS], [Sic], [Pie], [BW1], [Awo], and [Mac]; the first and second are suited for general audiences, the third and fourth are suited for computer scientists, and the fifth and sixth are suited for mathematicians (in each class the first reference is easier than the second).

A basic olog, defined in Section 2, is a category in which the objects and arrows have been labeled by English-language phrases that indicate their intended meaning. The objects represent types of things, the arrows represent functional relationships (also known as aspects, attributes, or observables), and the commutative diagrams represent facts. Here is a simple olog about an amino acid called arginine ([W1]):

(1)



The idea of representing information in a graph is not new. For example the Resource Descriptive Framework (RDF) is a system for doing just that [CM]. The key difference between a category and a graph is the consideration of paths, and that two paths from A to B may be declared identical in a category (see [Sp3]). For example, we can further declare that in Figure (1), the diagram

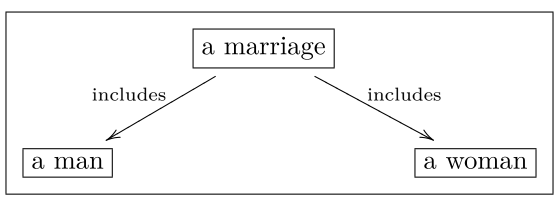
$$\begin{array}{ccc} A & \longrightarrow & E \\ \downarrow & & \downarrow \\ X & \longrightarrow & R \end{array} \tag{2}$$

commutes, i.e. that the two paths $A \rightrightarrows R$ are equivalent, which can be translated as follows. Let A be a molecule of arginine. On the one hand A, being an amino acid has a side chain; on the other hand A has an electrically-charged side-chain, which is of course a side chain. We seem to have associated *two* side-chains to A, but in fact they both refer to the same physical thing, the same side-chain. Thus, the two paths $A \to R$ are deemed equivalent. The fact that this equivalence may seem trivial is not an indictment of the category idea but instead reinforces its importance — we must be able to indicate the obvious in a given situation because the obvious is the most essential.

While many situations can be modeled using basic ologs (categories), we often need to encode more structure. For this we will need so-called sketches. An olog will be defined as a finite limit, finite colimit sketch (see [BW2]), meaning we have the ability to encode objects ("types"), arrows ("aspects"), commutative diagrams ("facts"), as well as finite limits ("layouts") and finite colimits ("groupings").

Throughout this paper, whenever I refer to "the author" of an olog I am referring to the fictitious person who created it. I will refer to myself, David Spivak, author of the present paper, as "I" so as not to confuse things. Whenever I say "we", I mean myself as well as anyone who is reading this paper or thinking about what ologs mean, how they are constructed, or how they may be useful.

Warning 1.0.1. The author of an olog has a world-view, some fragment of which is captured in the olog. When person A examines the olog of person B, person A may or may not "agree with it." For example, person B may have the following olog



which associates to each marriage a man and a woman. Person A may take the position that some marriages involve two men or two women, and thus see B's olog as "wrong." Such disputes are not "problems" with either A's olog or B's olog, they are discrepancies between world-views. Hence, throughout this paper, a reader R may see a displayed olog and notice a discrepancy between R's world-view and my own, but R should not worry that this is a problem. This is not to say that ologs need not follow rules, but instead that the rules are enforced to ensure that an olog is structurally sound, rather than that it "correctly reflects reality," whatever that may mean.

1.1. **Plan of this paper.** In this paper, I will define ologs and give several examples. I will state some rules of "good practice" which help one to author ologs that are meaningful to others and easily extendable. I will begin in Section 2 by laying out the basics: types as objects, aspects as arrows, and facts as commutative diagrams. In Section 3, I will explain how to attach "instance" data to an olog and hence realize ologs as database schemas. In Section 4, I will discuss meaningful connections between ologs (via functors) and the higher-dimensional web of information that may develop from such linkages. In Sections 5 and 6, I will extend the olog definition language to include "layouts" and "groupings", which make for more expressive ologs; I will also describe two applications, one which explicates the computation of the factorial function, and the other which defines a notion from pure mathematics (that of pseudo-metric spaces). Finally, in Section 7, I will discuss some possible directions for future research.

For the remainder of the present section, I will explain how ologs relate to existing ideas in the field of knowledge representation.

1.2. **The semantic advantage of ologs: modularity.** The difference between ologs and prose is modularity: small conceptual pieces can form large ideas, and these pieces work best when they are reusable. The same phenomenon is true throughout computer science and mathematics. In programming languages, modularity brings not only vast efficiency to the writing of programs but enables an "abstraction barrier" that keeps the ideas clean. In mathematics, the most powerful results are often simple lemmas that are reusable in a wide variety of circumstances.

Web pages that consist of prose writing are often referred to as *information silos.* The idea is that a silo is a "big tube of stuff" which is not organized in any real way. Links between web pages provide some structure, but such a link does not carry with it a precise method to correlate the information within the two pages. In science, one author may reference another paper, but such a reference carries very little structure — it just points to a silo.

Ologs can be connected with links which are much richer than the link between two silos could possibly be. Individual concepts and connections within one olog can be "functorially aligned" with concepts and connections in another. A functor creates a precise connection between the work of one author to the work of another so that the precise nature of the comparison is not left to the reader's imagination but explicitly specified. The ability to incorporate mathematical precision into the sharing of ideas is a central feature of ologs.

1.3. **Relation to other models.** There are many languages for knowledge representation (KR). For example, there are database languages such as SQL, ontology languages such as RDF and OWL, the language of Semantic Nets, and others (see [Bor]). One may ask what makes the olog concept different or better than the others.

The first response is that ologs are closely related to the above ideas. Indeed, all of these KR models can be "categorified" (i.e. phrased in the language of category theory) and related by functors, so that many of the ideas align and can be transferred between the different systems. In fact, as I will make clear in Section 3, ologs are almost identical to the categorical model of databases presented in [Sp2].

However, ologs have advantages over many existing KR models. The first advantage arises from the notion of commutative diagrams (which allow us to equate different paths through the domain, see Section 2.3) and of limits and colimits (which allow us to lay out and group things, see Sections 5 and 6). The additional expressivity of ologs give them a certain semantic clarity and interoperability that cannot be achieved with graphs and networks in the usual sense. The second advantage arises from the notion of functors which allow us to connect together different ologs in a meaningful way, enabling sharing and "data fusion." This will be discussed more in Section 4.

In the remainder of this section I will provide a few more details on the relationship between ologs and each of the above KR models: databases, RDF/OWL, and semantic nets. The reader who does not know or care much about other systems of knowledge representation can skip to Section 1.4.

1.3.1. *Ologs and Databases.* A database is a system of tables, each table of which consists of a header of columns and a set of rows. A table represents a type of thing T, each column represents an attribute of T, and each row represents an example of T. An attribute is itself a "type of thing", so each column of a table points to another table.

The relationship between ologs and databases is that every box B in an olog represents a type of thing and every arrow $B \to X$ emanating from B represents an attribute of B (whose results are of type X). Thus the boxes and arrows in an olog correspond to tables and their columns in a database. The rows of each table in a database will correspond to "instances" of each type in an olog. Again, this will be made more clear in Section 3 or one can see [Sp2].

The point is that every olog can serve as a database schema, and the schemas represented by ologs range from simple (just objects and arrows) to complex (including commutative diagrams, products, sums, etc.). However, whereas database schemas are often prescriptive ("you must put your data into this format!"), ologs are usuall descriptive ("this is how I see things"). One can think of an olog as an interface between people and databases: an olog is human readable, but it is also easily converted to a database schema upon which powerful applications can be put to work.

1.3.2. *Ologs and RDF / OWL.* In [Sp2], I explained how a categorical database can be converted into an RDF triple store using the Grothendieck construction. The main difference between a categorical database schema (or an olog) and an RDF schema is that one cannot specify commutativity in an RDF schema. Thus one cannot express things like "the woman parent of a person x is the mother of x." Without this expressivity, it is hard to enforce much rigor, and thus RDF data tends to be too loose for many applications.

To rectify this, one can use OWL schemas to express many more constraints on classes and properties. While one can express almost anything in OWL (it is a second-order language), this expressivity comes at a cost. As we will see with ologs, one can choose between different levels of expressivity in the OWL-hierarchy. While the full version of OWL may be slightly more powerful than an olog, if one includes the power of category theory that sits behind ologs, they are probably quite similar. For experts, I can give the example that reification in OWL or RDF can be accomplished by taking the Grothendieck construction of the RDF graph considered as a presheaf on the category $\bullet \rightrightarrows \bullet$. Examples such as this can lead to a more precise comparison.

However, OWL is written in XML, a language that is difficult for humans to read. There are several different proposals for how to ameliorate this problem, but none so far is based on category theory, and hence they lack a certain kind of formal grounding. As I have said, the purpose of ologs is to bridge the gap between human readability and computer readability, to give humans the ability to record ideas in a precise way without being experts in a computer language, while resting on a solid foundation of category theory.

1.3.3. *Semantic Nets.* On the surface, ologs look the most like semantic networks, or concept webs, but there are important differences between the two notions. First, arrows in a semantic network need not indicate functions; they can be relations. So there could be an arrow $\ulcorner$a father$\urcorner \xrightarrow{\text{has}} \ulcorner$a child$\urcorner$ in a semantic network, but not in an olog (see Section 2.2.3 for how the same idea is expressible in an olog). There is a nice category of sets and relations, often denoted **Rel**, but this category is harder to reason about than the ordinary category of sets and functions (often denoted **Set**) is. Thus, as mentioned above, semantic networks are categorifiable (using **Rel**), but this underlying formalism does not appear to play a part in the study or use of semantic networks. However, some attempt to integrate category theory and neural nets has been made, see [HC].

Moreover, commutative diagrams and other expressive abilities held by ologs are not generally part of the semantic network concept (see [Sow]). For these reasons, semantic networks tend to be brittle: minor changes can have devastating effects. For example, if two semantic networks are somehow synced up and then one is changed, the linkage must be revised or may be altogether broken. Such a disaster is often avoided if one uses categories: because different paths can be equivalent, one can simply add new ideas (types and aspects) without changing the semantic meaning of what was already there.

1.4. **Acknowledgements.** I would like to thank Mathieu Anel and Henrik Forsell for many pleasant and quite useful conversations. I would also like to thank Micha Breakstone for his help on understanding the relationship between ologs and linguistics. Finally I would like to thank Dave Balaban for helpful suggestions on this document itself.

2. Types, aspects, and facts

In this section I will explain basic ologs, which involve types, aspects, and facts. A basic olog is a category in which each object and arrow has been labeled by text; throughout this paper we will assume that text to be written in English.

The purpose of this section is to show how one can convert a real-world situation into an olog. It is probably impossible to explain this process precisely in words. Instead, I will explain mainly by example. I will give "rules of good practice" that lead to good ologs. While these rules are not strictly necessary, they help to ensure that the olog is properly formulated. As the Dalai Lama says "Learn the rules so you know how to break them properly."

2.1. **Types.** A type is an abstract concept, a distinction the author has made. We represent each type as a box containing a *singular indefinite noun phrase.* Each of the following four boxes is a type:

(3)

a man	an automobile
a pair (a, w), where w is a woman and a is an automobile	a pair (a, w) where w is a woman and a is a blue automobile owned by w

Each of the four boxes in (3) represents a type of thing, a whole class of things, and the label on that box is what one should call *each example* of that class. Thus ⌜a man⌝ does not represent a single man, but the set of men, each example of which is called "a man". Similarly, the bottom right box represents an abstract type of thing, which probably has more than a million examples, but the label on the box indicates the common name for each such example.

Typographical problems emerge when writing a text-box in a line of text, e.g. the text-box [a man] seems out of place here, and the more in-line text-boxes there are, the worse it gets. To remedy this, I will denote types which occur in a line of text with corner-symbols, e.g. I will write ⌜a man⌝ instead of [a man].

2.1.1. *Types with compound structures.* Many types have compound structures; i.e. they are composed of smaller units. Examples include

(4)

a man and a woman	a food f and a child c such that c ate all of f	a triple (p, a, j) where p is a paper, a is an author of p, and j is a journal in which p was published

It is good practice to declare the variables in a "compound type", as I did in the last two cases of (4). In other words, it is preferable to replace the first box above with something like

a man m and a woman w	or	a pair (m, w) where m is a man and w is a woman

so that the variables (m, w) are clear.

Rules of good practice 2.1.1. A type is presented as a text box. The text in that box should

(i) begin with the word "a" or "an";
(ii) refer to a distinction made and recognizable by the author;
(iii) refer to a distinction for which instances can be documented;
(iv) not end in a punctuation mark;
(v) declare all variables in a compound structure.

The first, second, and third rules ensure that the class of things represented by each box appears to the author as a well-defined set; see Section 3 for more details. The fourth and fifth rules encourage good "readability" of arrows, as will be discussed next in Section 2.2.

I will not always follow the rules of good practice throughout this document. I think of these rules being followed "in the background" but that I have "nicknamed" various boxes. So $\ulcorner$Steve$\urcorner$ may stand as a nickname for $\ulcorner$a thing classified as Steve$\urcorner$ and $\ulcorner$arginine$\urcorner$ as a nickname for $\ulcorner$a molecule of arginine$\urcorner$.

2.2. **Aspects.** An aspect of a thing x is a way of viewing it, a particular way in which x can be regarded or measured. For example, a woman can be regarded as a person; hence "being a person" is an aspect of a woman. A man has a height (say, taken in inches), so "having a height (in inches)" is an aspect of a man. In an olog, an aspect of A is represented by an arrow $A \to B$, where B is the set of possible "answers" or results of the measurement. For example when observing the height of a man, the set of possible results is the set of integers, or perhaps the set of integers between 20 and 120.

$$\boxed{\text{a woman}} \xrightarrow{\text{is}} \boxed{\text{a person}} \tag{5}$$

$$\boxed{\text{a man}} \xrightarrow{\text{has as height (in inches)}} \boxed{\text{an integer between 20 and 120}} \tag{6}$$

I will formalize the notion of aspect by saying that aspects are functional relationships. Suppose we wish to say that a thing classified as X has an aspect f whose result set is Y. This means there is a functional relationship called f between X and Y, which can be denoted $f\colon X \to Y$. We call X the *domain of definition* for the aspect f, and we call Y the *set of values* for f. For example, a man has a height in inches whose result set is the set of integers, and we could denote this by $h\colon M \to \mathbf{Int}$. Here, M is the domain of definition for height and **Int** is the set of values.

A set may always be drawn as a blob with dots in it. If X and Y are two sets, then a *a function from X to Y*, denoted $f\colon X \to Y$ can be presented by drawing arrows from dots in blob X to dots in blob Y. There are two rules:

(i) each arrow must point *from* a dot in X *to* a dot in Y;
(ii) each dot in X must have precisely *one* arrow emanating from it.

Given an element $x \in X$, the arrow emanating from it points to some element $y \in Y$, which we call *the image of x under f* and denote $f(x) = y$.

Again, in an olog, an aspect of a thing X is drawn as a labeled arrow pointing from X to a "set of values." Let us concentrate briefly on the arrow in (5). The domain of definition is the set of women (a set with perhaps 3 billion elements); the set of values is the set of persons (a set with perhaps 6 billion elements). We can imagine drawing an arrow from each dot in the "woman" set to a unique dot in the "person" set. No woman points to two different people, nor to zero people — each woman is exactly one person — so the rules for a functional relationship are satisfied. Let us now concentrate briefly on the arrow in (6). The domain of definition is the set of men, the set of values is the set of integers $\{20, 21, 22, \ldots, 119, 120\}$. We can imagine drawing an arrow from each dot in the "man" set to a single dot in the "integer" set. No man points to two different heights, nor can a man have no height: each man has exactly one height. Note however that two different men can point to the same height.

2.2.1. *Invalid aspects.* I tried above to clarify what it is that makes an aspect "valid", namely that it must be a "functional relationship." In this subsection I will show two arrows which on their face may appear to be aspects, but which on closer inspection are not functional (and hence are not valid as aspects).

Consider the following two arrows:

(7*) $$\boxed{\text{a person}} \xrightarrow{\text{has}} \boxed{\text{a child}}$$

(8*) $$\boxed{\text{a mechanical pencil}} \xrightarrow{\text{uses}} \boxed{\text{a piece of lead}}$$

A person may have no children or may have more than one child, so the first arrow is invalid: it is not functional because it does not satisfy rule (2) above. Similarly, if we drew an arrow from each mechanical pencil to each piece of lead it uses, it would not satisfy rule (2) above.

Of course, in keeping with Warning 1.0.1, the above arrows may not be wrong but simply reflect that the author has a strange world-view or a strange vocabulary. Maybe the author believes that every mechanical pencil uses exactly one piece of lead. If this is so, then ⌜a mechanical pencil⌝ $\xrightarrow{\text{uses}}$ ⌜a piece of lead⌝ is indeed a valid aspect! Similarly, suppose the author meant to say that each person *was once* a child, or that a person has an inner child. Since every person has one and only one inner child (according to the author), the map ⌜a person⌝ $\xrightarrow{\text{has as inner child}}$ ⌜a child⌝ is a valid aspect. We cannot fault the author for such a view, but note that we have changed the name of the label to make its intention more explicit.

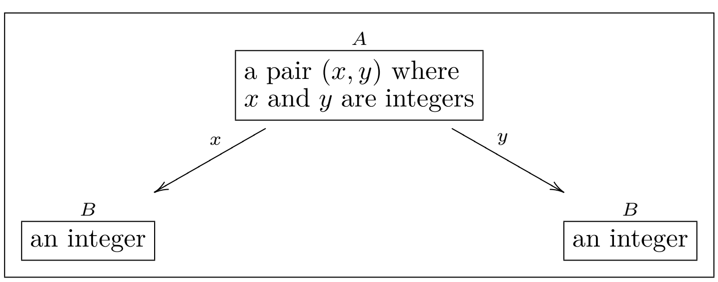
2.2.2. *Reading aspects and paths as English phrases.* Each arrow (aspect) $X \xrightarrow{f} Y$ can be read by first reading the label on its source box (domain of definition) X, then the label on the arrow f, and finally the label on its target box (set of values) Y. For example, the arrow

(9) $$\boxed{\text{a book}} \xrightarrow{\text{has as first author}} \boxed{\text{a person}}$$

is read "a book has as first author a person".

Sometimes the label on an arrow can be shortened or dropped altogether if it is obvious from context. We will discuss this more in Section 2.3 but here is a common example from the way I write ologs.

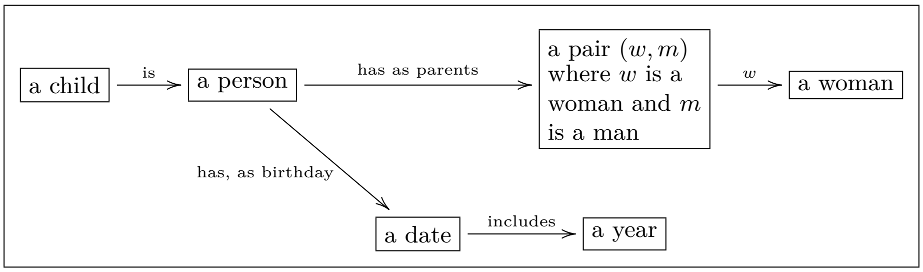
(10)



Neither arrow is readable by the protocol given above (e.g. "a pair (x, y) where x and y are integers x an integer" is not an English sentence), and yet it is obvious what each map means. For example, given $(8, 11)$ in A, arrow x would yield 8 in box B. The label x can be thought of as a nickname for the full name "yields, via the value of x," and similarly for y. I do not generally use the full name for fear that the olog would become cluttered with text.

One can also read paths through an olog by inserting the word "which" after each intermediate box. For example the following olog has two paths of length 3 (counting arrows in a chain):

(11)



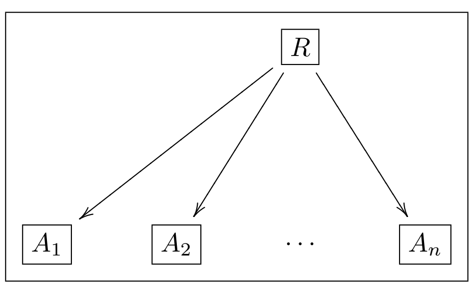
The top path is read "a child is a person, which has as parents a pair (w, m) where w is a woman and m is a man, which yields, via the value of w, a woman." The reader should read and understand the content of the bottom path.

2.2.3. *Converting non-functional relationships to aspects.* There are many relationships that are not functional, and these cannot be considered aspects. Often the word "has" indicates a relationship — sometimes it is functional as in ⌜a person⌝ $\xrightarrow{\text{has}}$ ⌜a stomach⌝, and sometimes it is not, as in ⌜a father⌝ $\xrightarrow{\text{has}}$ ⌜a child⌝. (Obviously, a father may have more than one child.) A quick fix would be to replace the latter by ⌜a father⌝ $\xrightarrow{\text{has}}$ ⌜a set of children⌝. This is ok, but the relationship between ⌜a child⌝ and ⌜a set of children⌝ then becomes an issue to deal with. There is another way to indicate such "non-functional" relationships.

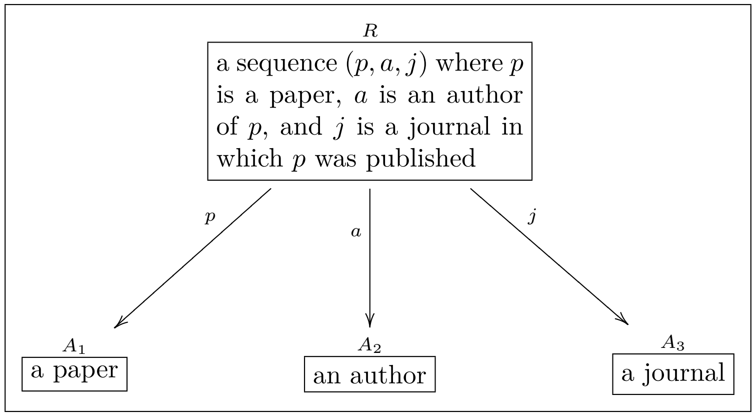
In mathematics, a relation between sets A_1, A_2, and so on through A_n is defined to be a subset of the Cartesian product

$$R \subseteq A_1 \times A_2 \times \cdots \times A_n.$$

The set R represents those sequences $(a_1, a_2, \ldots, a_n)$ that are so-related. In an olog, we represent this as follows



For example,



Whereas $A_1 \times A_2 \times A_3$ includes all possible triples (p, a, j) where a is a person, p is a paper, and j is a journal, it is obvious that not all such triples are found in R. Thus R represents a proper subset of $A_1 \times A_2 \times A_3$.

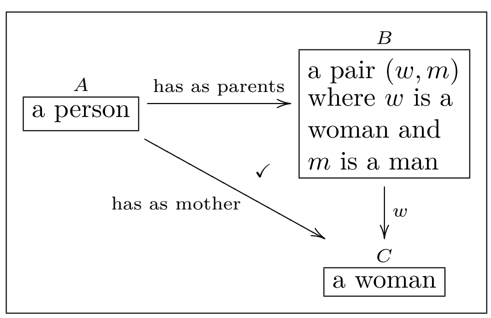
Rules of good practice 2.2.1. An aspect is presented as a labeled arrow, pointing from a source box to a target box. The arrow text should

(i) begin with a verb;
(ii) yield an English sentence, when the source-box text followed by the arrow text followed by the target-box text is read;
(iii) refer to a functional dependence: each instance of the source type should give rise to a specific instance of the target type;

2.3. **Facts.** In this section I will discuss facts and their relationship to "path equivalences." It is such path equivalences, which exist in categories but do not exist in graphs, that make category theory so powerful. See [Sp3] for details.

Given an olog, the author may want to declare that two paths are equivalent. For example consider the two paths from A to C in the olog

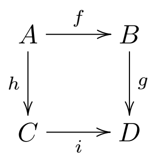
(12)



We know as English speakers that a woman parent is called a mother, so these two paths $A \to C$ should be equivalent. A more mathematical way to say this is that the triangle in Olog (12) *commutes.*

A *commutative diagram* is a graph with some declared path equivalences. In the example above we concisely say "a woman parent is equivalent to a mother." We declare this by defining the diagonal map in (12) to be *the composition* of the horizontal map and the vertical map.

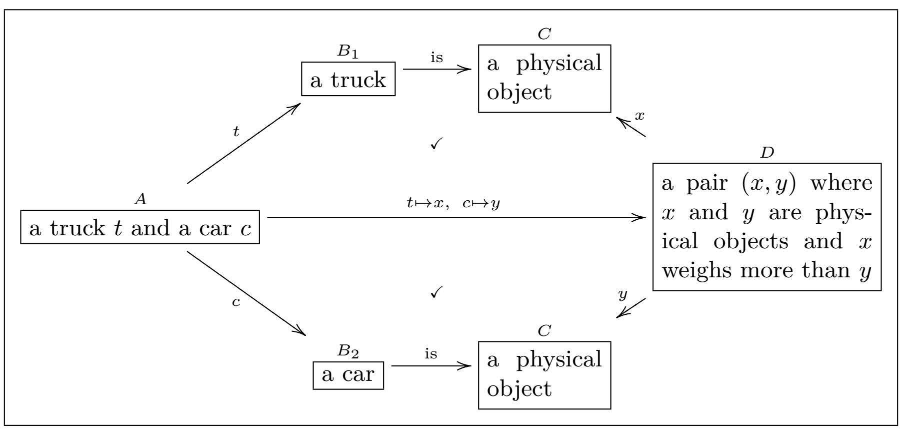
I generally prefer to indicate a commutative diagram by drawing a check-mark, ✓, in the region bounded by the two paths, as in Olog (12). Sometimes, however, one cannot do this unambiguously on the 2-dimensional page. In such a case I will indicate the commutative diagrams (fact) by writing an equation. For example to say that the diagram



commutes, we could either draw a checkmark inside the square or write the equation $f; g = h; i$ above it. Either way, it means that "f then g" is equivalent to "h then i".

2.3.1. *More complex facts.* Recording real-world facts in an olog can require some creativity. Whereas a fact like "the brother of ones father is ones uncle" is recorded as a simple commutative diagram, others are not so simple. I will try to show the range of expressivity of commutative diagrams in the following two examples.

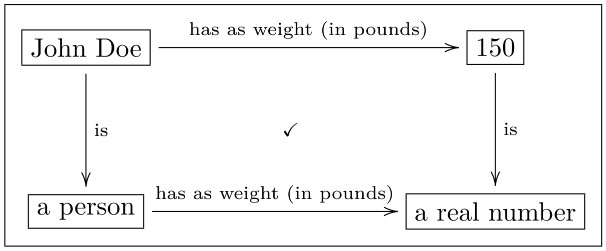
Example 2.3.2. How would one record a fact like "a truck weighs more than a car"? I suggest something like this:



where both top and bottom commute. This olog exemplifies the fact that simple sentences sometimes contain large amounts of information. While the long map may seem to suffice to convey the idea "a truck weighs more than a car," the path equivalences (declared by check-marks) serve to ground the idea in more basic types. These other types tend to be useful for other purposes, both within the olog and when connecting it to others.

2.3.3. *Specific facts at the schema level.* Another fact one might wish to record is that "John Doe's weight is 150 lbs." This is established by declaring that the following diagram commutes:

(13)



If one only had the top line, it would be less obvious how to connect its information with that of other ologs. (See Section 4 for more on connecting different ologs).

Note that the top line in Diagram (13) might also be considered as existing at the "data level" rather than at the "schema level." In other words, one could see John Doe as an "instance" of ⌜a person⌝, rather than as a type in and of itself, and similarly see 150 as an instance of ⌜a real number⌝. This idea of an olog having a "data level" is the subject of the Section 3.

Rules of good practice 2.3.4. A fact is the declaration that two paths (having the same source and target) in an olog are equivalent. Such a fact is either presented as a checkmark between the two paths (if such a check-mark is unambiguous) or by an equation. Every such equivalence should be declared.

3. Instances

The reader at this point hopefully sees an olog as a kind of "concept map," and it is one, albeit a concept map with a formal structure (implicitly coming from category theory) and specific rules of good practice. In this section I will show that one can also load an olog with data. Each type can be assigned a set of instances, each aspect will map the instances of one type to instances of the other, and each fact will equate two such mappings. I give examples of these ideas in Section 3.1.

In Section 3.2, I will show that in fact every olog can also serve as the layout for a database. In other words, given an olog one can immediately generate a *database schema*, i.e. a system of tables, in any reasonable data definition language such as that of SQL. The tables in this database will be in one-to-one correspondence with the types in the olog. The columns of a given table will be the aspects of the corresponding type, i.e. the arrows whose source is that type. Commutative diagrams in the olog will give constraints on the data.

In fact, this idea is my basic thesis in [Sp2], even though the word olog does not appear in that paper. There I instead explained that a category $\mathcal{C}$ naturally can be viewed as a database schema and that a functor $I\colon \mathcal{C} \to \mathbf{Set}$, where **Set** is the category of sets, is a database state. Since an olog is a drawing of a category, it is also a drawing of a database schema. The current section is about the "states" of an olog, i.e. the kinds of data that can be captured by it.

3.1. **Instances of types, aspects, and facts.** Recall from Section 2 that basic ologs consist of types, displayed as boxes; aspects, displayed as arrows; and facts, displayed as equations or check-marks. In this section we discuss the instances of these three basic constructions. The rules of good practice (2.1.1, 2.2.1, and 2.3.4) were specifically designed to simplify the process of finding instances.

3.1.1. *Instances of types.* According to Rules 2.1.1, each box in an olog contains text which should refer to **a distinction made and recognizable by the author for which instances can be documented.** For example if my olog contains a box

a pair (p, c) where p is a person, c is a cat, and p has petted c

then I must have some concept of when this situation occurs. Every time I witness a new person-cat petting, I document it. Whether this is done in my mind, in a ledger notebook, or on a computer does not matter; however using a computer would probably be the most self-explanatory. Imagine a computer program in which one can create text boxes. Clicking a text box results in it "opening up" to show a list of documented instances of that type. If one is reading the CBS news olog and clicks on the box ⌜an episode of 60 Minutes⌝, he or she should see a list of all episodes of the TV show "60 Minutes."

3.1.2. *Instances of aspects.* According to Rules 2.2.1, each arrow in an olog should be labeled with text that refers to a functional relationship between the source box and the target box. A functional relationship $f\colon A \to B$ between finite sets A and B can always be written as a 2-column table: the first column is filled with the instances of type A and the second column is filled with their f-values, which are instances of type B.

For example, consider the aspect

$$\boxed{\text{a moon}} \xrightarrow{\text{orbits}} \boxed{\text{a planet}} \tag{14}$$

We can document some instances of this relationship using the following table:

(15)

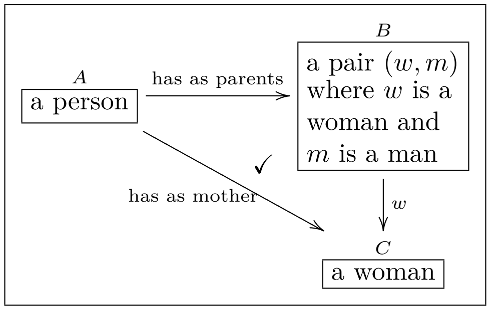
orbits	
a moon	**a planet**
The Moon	Earth
Phobos	Mars
Deimos	Mars
Ganymede	Jupiter
Titan	Saturn

Clearly, this table of instances can be updated as more moons are discovered by the author (be it by telescope, conversation, or research).

The correspondence between aspect (14) and Table (15) makes it clear that ologs can serve to hold data which exemplifies the author's worldview. In Section 3.2, I will show that ologs (which have many aspects and facts) can serve as bona fide database schemas.

3.1.3. *Instances of facts.* Recall the following olog:

(12)



and consider the following instances of the three aspects in it:

(16)

has as parents	
a person	**a pair (w, m) ...**
Cain	(Eve, Adam)
Abel	(Eve, Adam)
Chelsey	(Hillary, Bill)

w	
a pair (w, m) ...	**a woman**
(Eve, Adam)	Eve
(Hillary, Bill)	Hillary
(Margaret, Samuel)	Margaret
(Emily, Kris)	Emily

has as mother	
a person	**a woman**
Cain	Eve
Abel	Eve
Chelsey	Hillary

When we declare that the diagram in (12) commutes (using the check-mark), we are saying that for every instance of ⌜a person⌝ (of which we have three: Cain, Abel, and Chelsey), the two paths to ⌜a woman⌝ give the same answers. Indeed, for Cain the two paths are:

(i) Cain $\mapsto$ (Eve, Adam) $\mapsto$ Eve;
(ii) Cain $\mapsto$ Eve;

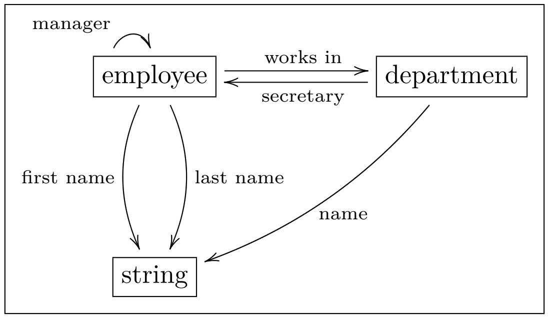
and these answers agree. If one changed any instance of the word “Eve” to the word “Steve” in one of the tables in (16), some pair of paths would fail to agree. Thus the “fact” that the diagram in (12) commutes ensures that there is some internal consistency between the meaning of parents and the meaning of mother, and this consistency must be born out at the instance level.

All of this will be formalized in Section 3.2.2.

3.2. **The relationship between ologs and databases.** Recall from Section 3.1.1 that we can imagine creating an olog on a computer. The user creates boxes, arrows, and compositions, hence creating a category $\mathcal{C}$. Each text-box x in the olog can be “clicked” by the computer mouse, an action which allows the user to “view the contents” of x. The result will be a set of things, which we might call $I(x) \in \mathbf{Set}$, whose elements are things of type x. So clicking on the box ⌜a man⌝ one sees $I($⌜a man⌝$)$, the set of everything the author has documented as being a man. For each aspect $f\colon x \to y$ of x, the user can see a function from the set $I(x)$ to $I(y)$, perhaps as a 2-column table as in (16).

The type x may have many aspects, which we can put together into a single multi-column table. Its columns are the aspects of x, and its rows are the elements of $I(x)$. Consider the following olog, taken from [Sp2] where it was presented as a database schema.

(17)



The type ⌜Employee⌝ has four aspects, namely `manager` (valued in ⌜Employee⌝), `works in` (valued in ⌜department⌝), and `first name` and `last name` (valued in ⌜string⌝). As a database, each type together with its aspects form a multi-column table, as in the following example.

Example 3.2.1. We can convert Olog (17) into a database schema. Each box represents a table, each arrow out of a box represents a column of that table. Here is an example state of that database.

(18)

employee				
Id	**first name**	**last name**	**manager**	**works in**
101	David	Hilbert	103	q10
102	Bertrand	Russell	102	x02
103	Alan	Turing	103	q10

department		
Id	**name**	**secretary**
q10	Sales	101
x02	Production	102

string
Id
a
b
⋮
z
aa
ab
⋮

Note that every arrow $f\colon x \to y$ of Olog (17) is represented in Database (18) as a column of table x, and that every cell in that column can be found in the Id column of table y. For example, every cell in the "works in" column of table **employee** can be found in the Id column of table **department**.

The point is that ologs can be drawn to represent a world-view (as in Section 2), but they can also store data. Rules 1,2, and 3 in 2.1.1 align the construction of an olog with the ability to document instances for each of its types.

3.2.2. *Instance data as a set-valued functor.* Let $\mathcal{C}$ be an olog. Section 3 so far has described instances of types, aspects, and facts and how all of these come together into a set of interconnected tables. The assignment of a set of instances to each type and a function to each aspect in $\mathcal{C}$, such that the declared facts hold, is called an assignment of *instance data* for $\mathcal{C}$. More precisely, instance data on $\mathcal{C}$ is a functor $\mathcal{C} \to \mathbf{Set}$, as in Definition 3.2.3.

Definition 3.2.3. Let $\mathcal{C}$ be a category (olog), and let **Set** denote the category of sets. An *instance of* $\mathcal{C}$ (or *an assignment of instance data for* $\mathcal{C}$) is a functor $I\colon \mathcal{C} \to \mathbf{Set}$. That is, it consists of

- a set $I(x)$ for each object (type) x in $\mathcal{C}$
- a function $I(f)\colon I(x) \to I(y)$ for each arrow (aspect) $f\colon x \to y$ in $\mathcal{C}$, and
- for each path-equivalence (fact)
$$f_1; f_2; \cdots ; f_n = g_1; g_2; \cdots ; g_m$$
declared in $\mathcal{C}$, an equality of functions
$$I(f_1); I(f_2); \cdots ; I(f_n) = I(g_1); I(g_2); \cdots ; I(g_m).$$

For more on this viewpoint of categories and functors, the reader can consult [Sp3].

4. Communication between ologs

Different people naturally have different world-views — we all have our own perspective on the world. However, we can only communicate with others when there is some commonality between our world-views. It is this commonality which allows us to explain our thoughts and perceptions to the other. In this section I will discuss how to formulate these channels of communication using ologs.

The mathematical concept that makes it all work is that of a *functor*. A functor is a mapping from one category to another that preserves all the declared structure. Whereas in Definition 3.2.3 I defined a functor from an olog to **Set**, here we will be discussing functors from one olog to another.

Typically if $\mathcal{C}$ and $\mathcal{D}$ are ologs, we cannot always find a meaningful functor from $\mathcal{C}$ to $\mathcal{D}$ because neither world-view will completely contain the other. As briefly mentioned in the paragraph above, we instead look for a "commonality" world-view $\mathcal{B}$ and functors $\mathcal{B} \to \mathcal{C}$ and $\mathcal{B} \to \mathcal{D}$, which together build the bridge between $\mathcal{C}$ and $\mathcal{D}$.

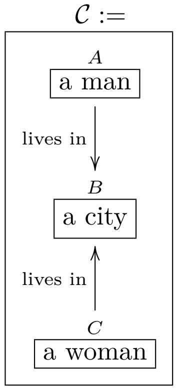
In this section we outline the idea of functors (see [Sp3] or [Awo] for a more rigorous presentation), and show how they can be used to connect ologs into an intricate web of different but overlapping world-views. This web consists of nodes, each of which corresponds to an olog, and connections between these nodes, i.e. an undirected graph. However, the result is generally more than just a graph: it can be given higher-dimensional structure by considering not just 2-way connections but also n-way connections for any positive integer n. For example a nuclear family of 4 often has a common vocabulary and understanding, constituting a 4-way overlap of world-views. The common understanding of the family could be recorded as a single "family olog." Some subset of the family could extend this olog in a way that some other members disagree with, hence forming a separate but connected olog. The resulting structures is called a symmetric simplicial set, a higher-dimensional generalization of an undirected graph.

In Section 4.1, I will describe functors and how they can be used to relate ologs and share data. In Section 4.2, I will describe the higher-dimensional network (symmetric simplicial set) that is formed when ologs are coauthored by groups of two or more individuals.

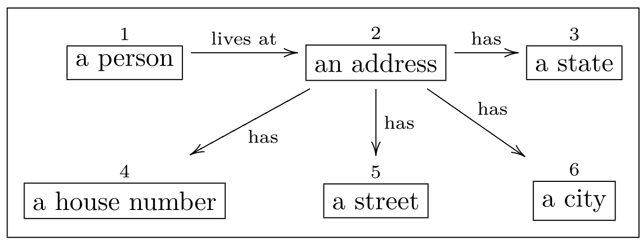
4.1. **Connecting ologs with functors.** Let $\mathcal{C}$ and $\mathcal{D}$ be basic ologs, as explained in Section 2. They consist of types (drawn as boxes), aspects (drawn as arrows), and facts (declared as equations or by check-marks). A *pre-functor F from $\mathcal{C}$ to $\mathcal{D}$*, denoted $F\colon \mathcal{C} \to \mathcal{D}$ is a mapping which sends types to types, aspects to aspects, and facts to facts. In other words, to define F one must specify for each type X in $\mathcal{C}$ a corresponding type $F(X)$ in $\mathcal{D}$. If there is an aspect $f\colon X \to Y$ in $\mathcal{C}$ one needs to specify a corresponding aspect $F(f)\colon F(X) \to F(Y)$ in $\mathcal{D}$. Finally, a fact such as $f; g = h$ which holds in $\mathcal{C}$ must also hold (as $F(f); F(g) = F(h)$) in $\mathcal{D}$. A *functor* is like a pre-functor except that an aspect in $\mathcal{C}$ may be sent to a path, i.e. a chain of aspects, in $\mathcal{D}$. Again, see [Sp3] for more on this.

Example 4.1.1. Let $\mathcal{C}$ and $\mathcal{D}$ respectively be the ologs

$\mathcal{C} :=$



$\mathcal{D} :=$



A functor from $\mathcal{C}$ to $\mathcal{D}$ is only required to send types to types and aspects to aspects (or to paths). For example, every type in $\mathcal{C}$ could be sent to 4=⌜a house number⌝ and every aspect in $\mathcal{C}$ could be sent to the identity path on 4. This is a legal functor, but it is not meaningful (how could we justify sending ⌜a man⌝ to ⌜a house number⌝?). While the rules of ologs keep some amount of order, they are no substitute for human common sense. In fact, the labels on the boxes and arrows help us find functors that will carry meaning.

Looking at the labels it becomes clear that the only functor $F\colon \mathcal{C} \to \mathcal{D}$ which is appropriate here is that which sends A and C to 1, and which sends B to 6. This functor sends each of the arrows in $\mathcal{C}$ to the chain $1 \xrightarrow{\text{lives at}} 2 \xrightarrow{\text{has}} 6$ in $\mathcal{D}$.

In Example 4.1.1 we saw that not every functor $F\colon \mathcal{C} \to \mathcal{D}$ between ologs is meaningful. One (roughly) says that F is *meaningful* if, for each type X in $\mathcal{C}$, every intended instance of X in $\mathcal{C}$ would be considered an instance of $F(X)$ by the author of $\mathcal{D}$ (in which case we say the intention for types is respected), and in a similar way the intention for aspects is respected. (For experts, if $I\colon \mathcal{C} \to \mathbf{Set}$ and $J\colon \mathcal{D} \to \mathbf{Set}$ are instance data for $\mathcal{C}$ and $\mathcal{D}$, then F is meaningful relative to I and J if one can exhibit a natural transformation $m\colon I \to F^*J$ as in [Sp2]).

One (roughly) says that F is *strongly meaningful* if it is meaningful and moreover, for each type X, the intended instances of X in $\mathcal{C}$ and the intended instances of $F(X)$ in $\mathcal{D}$ are the same. (For experts, we want m to be a natural isomorphism.)

Given a functor $F\colon \mathcal{C} \to \mathcal{D}$, there are rigorous ways of moving instance data on $\mathcal{C}$ to instance data on $\mathcal{D}$ and vice versa. In other words, once a connection is made between two ologs, the authors can share and related experiences with each other in their common language. These formulations are outside the scope of the present paper, but see [Sp2] for a detailed account.

4.1.2. *Common ground.* Given two world-views $\mathcal{C}$ and $\mathcal{D}$ there is little hope that one of them completely contains the other (even after allowing for renaming of types and aspects), and there is correspondingly little chance of finding a meaningful functor $\mathcal{C} \to \mathcal{D}$. Instead, the two parties should attempt to find a common ground, a third olog $\mathcal{B}$ and meaningful maps $F_C : \mathcal{B} \to \mathcal{C}$ and $F_D : \mathcal{B} \to \mathcal{D}$.

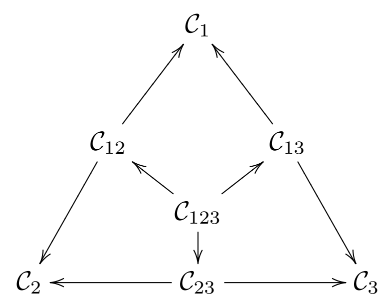
Such a common ground can be expanded and improved over time. The basic idea is that one party, say $\mathcal{C}$ can attempt to explain a new idea (type, aspect, or fact) to $\mathcal{D}$ in terms of the common ground. Then $\mathcal{D}$ can either interpret this idea as one he or she already has, learn from it (i.e. freely add it to his or her olog), or reject it. These ideas are formalized in purely categorical terms in [SA].

4.2. **A network of ologs.** In Section 4.1, I described how different ologs $\mathcal{C}$ and $\mathcal{D}$ can be connected through common ground $\mathcal{C} \leftarrow \mathcal{B} \to \mathcal{D}$. One should imagine a graph for which each node corresponds to an olog and each edge corresponds to a common-ground olog connecting its endpoints. This creates a 1-dimensional knowledge network.

However, as mentioned above, more than two people may share a common ground. For example a set of three companies that do business together may have a common-ground olog as part of a legal contract, or a set of 30 participants in a conference may have some common understanding of that subject. In fact, for any finite set of authors $X = \{A_1, A_2, \ldots, A_n\}$, there should be a common ground worldview (even if empty), say $\mathcal{C}_X$. If $Y \subseteq X$ is a subset, then there should be a map $\mathcal{C}_X \to \mathcal{C}_Y$ because any common understanding held by everyone in set X is held by everyone in subset Y also.

Example 4.2.1. When $n = 3$ we have:

(19)



This represents three individuals (1,2,3), their ologs, $(\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3)$, their pair-wise commonality ologs $(\mathcal{C}_{12}, \mathcal{C}_{13}, \mathcal{C}_{23})$, and their three-way commonality olog $(\mathcal{C}_{123})$.

This triangle, which stands for the interaction between persons 1,2,3 does not stand alone, but is part of an intricate web of other triangles and other such shapes. For example, persons 1 and 3 may be part of some different interacting group, say of persons 1,3,6,7, and hence the right edge of triangle (19) would be part of some tetrahedron with vertices 1,3,6,7.

Thus we have more than just a graph; we have what is known as a *symmetric simplicial set* as our network of ologs. These ideas are explained in detail in [Sp1]. We will return to this idea briefly in Section 7.

5. More expressive ologs I

In this section and the next (5 and 6) I will introduce limits and colimits within the context of ologs. These will allow authors to build ologs that are quite expressive. For example we can declare one type to be the union or intersection of other types.

The basic ologs discussed in previous sections are based on the mathematical notion of categories, whereas the olog presentation language we will discuss in this section and the next are based on *general sketches* (see [Mak]). The difference is in what can be expressed: in basic ologs we can declare types, aspects, and facts, whereas in general ologs we can express ideas like products and sums, as we will see below.

Many ideas can be expressed using only basic ologs, and when this is possible it is preferred. Many more ideas can be expressed with the addition of only "layouts" (corresponding to *finite limit sketches*), and again an author who can restrict himself or herself to this language will benefit for it. The reason is that when authors use a richer olog presentation language, the ability to compare different ologs becomes harder in two ways. First, it becomes harder to make meaningful connections (functors or sketch-maps) between different ologs, and second the set of theorems available for transferring instance data (see Section 3.2.2) from one olog to the other becomes restricted.

Regardless, these richer ologs will be the subject of this section and the next. I will begin by discussing layouts, which will be represented category-theoretically by "finite limits". As usual, the english terminology (layout) is not precise enough to express the notion I mean it to express (limit). Intuitively, a limit can be thought of as a system: it is a collection of units, each of a specific type, such that these units have compatible aspects. These will include types like ⌜a man and a woman with the same last name⌝. In Section 6 I will discuss groupings, which will be represented by colimits. These will include types like ⌜a thing that is either a pear or a watermelon⌝.

5.1. **Layouts.** One can define a layout as "a structured arrangement of items within certain limits; a plan for such arrangement." In other words, we can lay out or specify the need for a set of parts, each of a given type, such that the parts fit together well. This idea roughly corresponds to the notion of limits in category theory, especially limits in the category of sets. Given a diagram of sets and functions, the limit is the set of ways to accordingly choose one element from each. For example, we could have a type ⌜a car and a driver⌝, which category-theoretically is a product, but which we are calling a "layout" — a compound type whose parts are "laid out." Of course, the term layout is insufficient to express the precise meaning of limits, but it will have to do for now. To understand limits, one really only need understand pullbacks and products. These will be the subjects of Sections 5.2 and 5.3, or one can see [Awo] for more details.

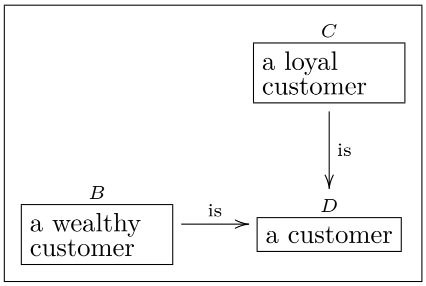
5.2. **Pullbacks.** Given three objects and two arrows arranged as to the left, the pullback is the commutative square to the right:

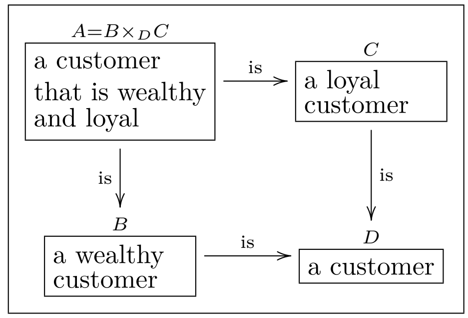
$$\text{Given:}\quad \begin{array}{ccc} & & C \\ & & \downarrow{\scriptstyle g} \\ B & \xrightarrow[f]{} & D \end{array} \qquad \text{the pullback is drawn:}\quad \begin{array}{ccc} A & \xrightarrow{f'} & C \\ {\scriptstyle g'}\downarrow & & \downarrow{\scriptstyle g} \\ B & \xrightarrow[f]{} & D. \end{array}$$

We write $A = B \times_D C$ and say "A is the pullback of B and C over D." The question is, what does it signify? I will begin with some examples and then give a precise definition.

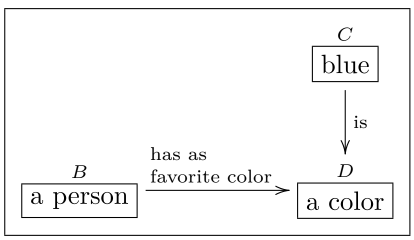
Example 5.2.1. I will now give four examples to motivate the definition of pullback. In the first example, (20), both B and C will be subtypes of D, and in such cases the pullback will be their intersection. In the next two examples (21 and 22), only B will be a subtype of D, and in such cases the pullback will be the "corresponding subtype of C" (as should make sense upon inspection). In the last example (23), neither B nor C will be a subtype of D. In each line below, the pullback of the diagram to the left is the diagram to the right. The reader should think of the left-hand olog as a kind of problem to which the new box A in the right-hand olog is a solution.

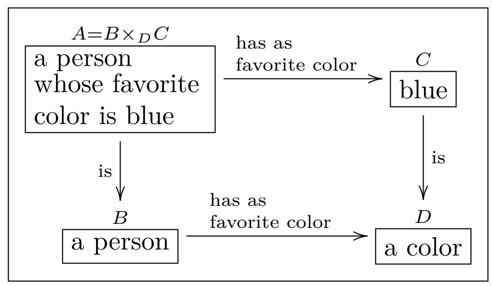
(20)


C
a loyal customer
is
B
a wealthy customer
is
D
a customer

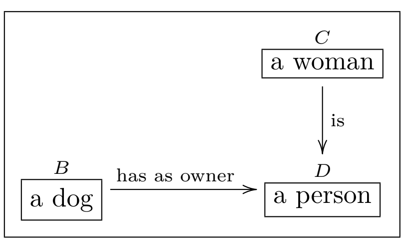

A=B×DC
a customer that is wealthy and loyal
is
C
a loyal customer
is
is
B
a wealthy customer
is
D
a customer

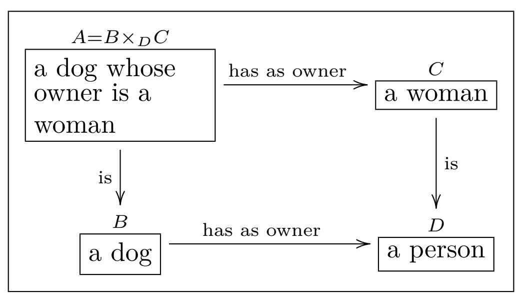
(21)


C
blue
is
B
a person
has as favorite color
D
a color

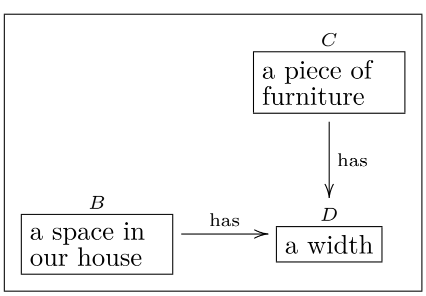

A=B×DC
a person whose favorite color is blue
has as favorite color
C
blue
is
is
B
a person
has as favorite color
D
a color

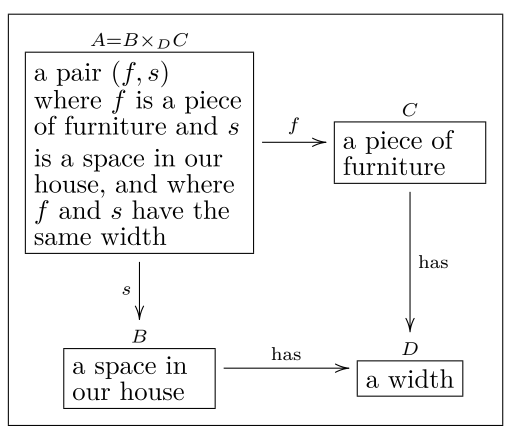
(22)


C
a woman
is
B
a dog
has as owner
D
a person


A=B×DC
a dog whose owner is a woman
has as owner
C
a woman
is
is
B
a dog
has as owner
D
a person

(23)


C
a piece of furniture
has
B
a space in our house
has
D
a width


A=B×DC
a pair (f, s) where f is a piece of furniture and s is a space in our house, and where f and s have the same width
f
C
a piece of furniture
s
has
B
a space in our house
has
D
a width

See Example 5.2.3 for a justification of these, in light of Definition 5.2.2.

The following is the definition of pullbacks in the category of sets. For an olog, the instance data are given by sets (at least in this paper, see Section 3), so this definition suffices for now. See [Awo] for more details on pullbacks.

Definition 5.2.2. Let B, C, and D be sets, and let $f\colon B \to D$ and $g\colon C \to D$ be functions. The *pullback* of $B \xrightarrow{f} D \xleftarrow{g} C$, denoted $B \times_D C$, is defined to be the set

$$B \times_D C := \{(b, c) \mid b \in B, c \in C, \text{ and } f(b) = g(c)\}$$

together with the obvious maps $B \times_D C \to B$ and $B \times_D C \to C$, which send an element (b, c) to b and to c, respectively. In other words, the pullback of $B \xrightarrow{f} D \xleftarrow{g} C$ is a commutative square

$$\begin{array}{ccc} B \times_D C & \longrightarrow & C \\ \downarrow & & \downarrow{\scriptstyle g} \\ B & \xrightarrow[f]{} & D. \end{array}$$

Example 5.2.3. In Example 5.2.1 I gave four examples of pullbacks. For each, I will consider $B \xrightarrow{f} D \xleftarrow{g} C$ to be sets and functions as in Definition 5.2.2 and explain how the set A follows that definition, i.e. how its label fits with the set $B \times_D C = \{(b, c) \mid b \in B, c \in C, \text{ and } f(b) = g(c)\}$.

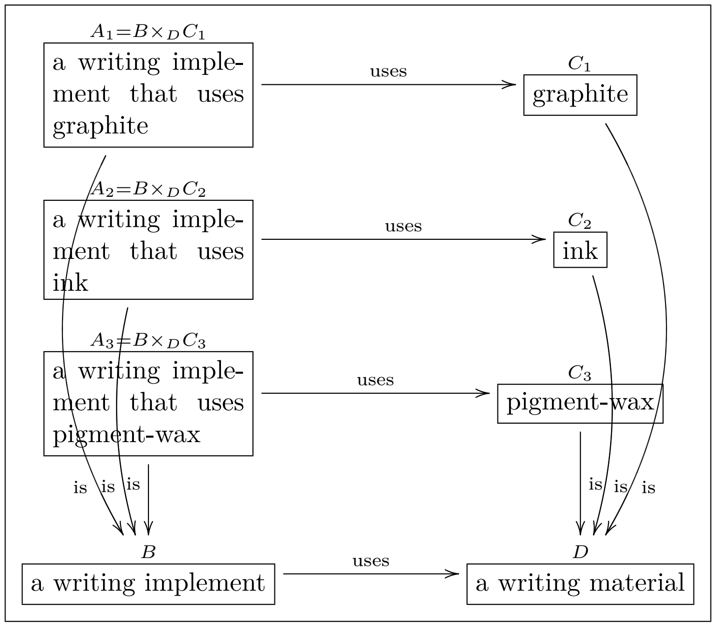
In the case of (20), the set $B \times_D C$ should consist of pairs (w, l) where w is a wealthy customer, l is a loyal customer, and w is equal to l (as customers). But if w and l are the same customer then (w, l) is just a customer that is both wealthy and loyal, not two different customers. In other words, an instance of the pullback is a customer that is both loyal and wealthy, so the label of A fits.

In the case of (21), the set $B \times_D C$ should consist of pairs (p, b) where p is a person, b is the color blue, and the favorite color of p is equal to b (as colors). In other words, it is a person whose favorite color is blue, so the label of A fits. If desired, one could instead label A with ⌜a pair (p, b) where p is a person, b is blue, and the favorite color of p is b⌝.

In the case of (22), the set $B \times_D C$ should consist of pairs (d, w) where d is a dog, w is a woman, and the owner of d is equal to w (as people). In other words, it is a dog whose owner is a woman, so the label of A fits. If desired, one could instead label A with ⌜a pair (d, w) where d is a dog, w is a woman, and the owner of d is w⌝.

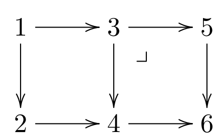
In the case of (23), the set $B \times_D C$ should consist of pairs (f, s) where f is a piece of furniture, s is a space in our house, and the width of f is equal to the width of s. This is fits perfectly with the label of A.

5.2.4. *Using pullbacks to classify.* To distinguish between two things, one must find a common aspect of the two things for which they have differing results. For example, a pen is different from a pencil in that they both use some material to write (a common aspect), but the two materials they use are different. Thus the material which a writing implement uses is an aspect of writing implements, and this aspect serves to segregate or classify them. I can think of three such writing-materials: graphite, ink, and pigment-wax. For each, I will make a layout in the olog below:



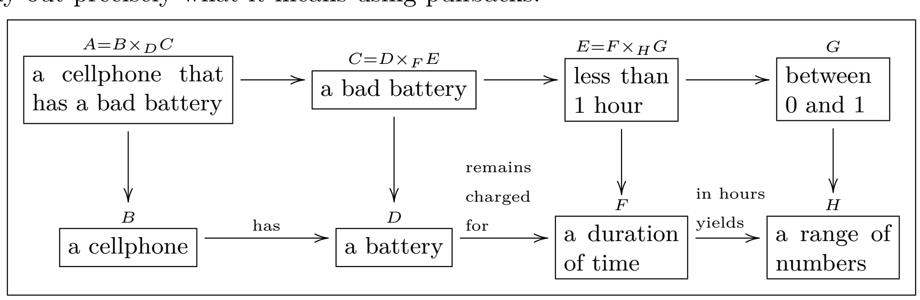
One could also replace the label of box A_1 with "a pencil", the label of box A_2 with "a pen", and the label of box A_3 with "a crayon"; in so doing, the layouts at the top would *define* a pencil, a pen, and a crayon to be a writing implement that uses respectively graphite, ink, and pigment-wax.

5.2.5. *Building pullbacks on pullbacks.* There is a theorem in category theory which states the following. Suppose given two commutative squares



such that the right-hand square (3,4,5,6) is a pullback. It follows that if the left-hand square (1,2,3,4) is a pullback then so is the big rectangle (1,2,5,6). It also follows that if the big rectangle (1,2,5,6) is a pullback then so is the left-hand square (1,2,3,4). This fact can be useful in authoring ologs.

For example, the type ⌜a cellphone that has a bad battery⌝ is vague, but we can lay out precisely what it means using pullbacks:



The category-theoretic fact described above says that since $A = B \times_D C$ and $C = D \times_F E$, it follows that $A = B \times_F E$. That is, we can decuce the definition "a cellphone that has a bad battery is defined as a cellphone that has a battery which remains charged for less than one hour." In other words, $A = B \times_F E$.

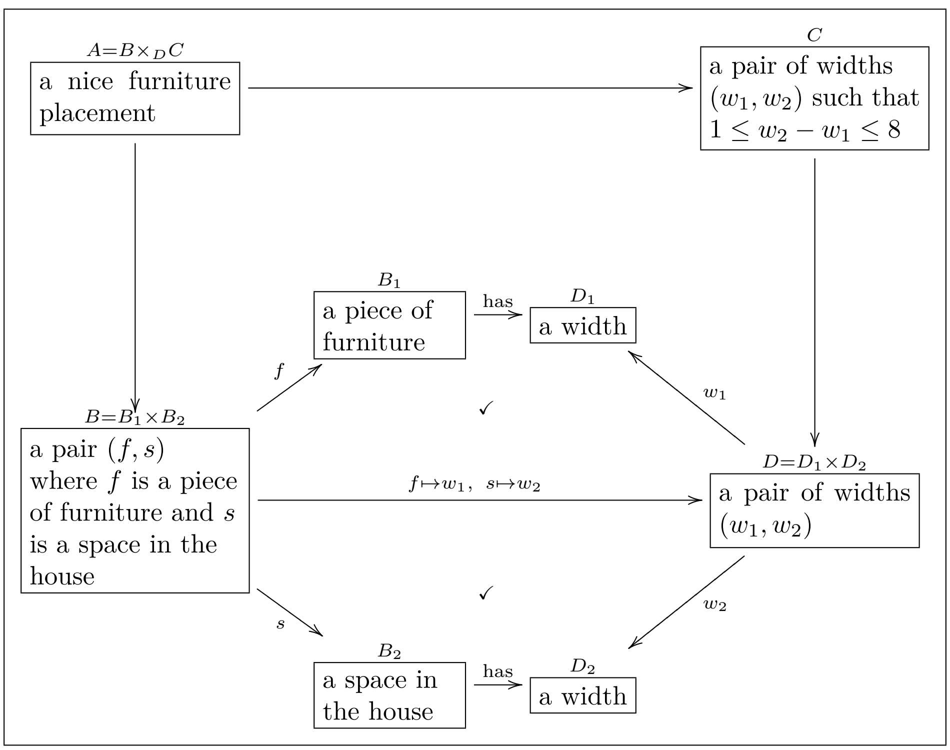
5.3. **Products.** Given a set of types (boxes) in an olog, one can select one instance from each. All the ways of doing just that comprise what is called the product of these types. For example, if $A =$ ⌜a number between 1 and 10⌝ and $B =$ ⌜a letter between x and z⌝, the product includes a total of 30 elements, including $(4, z)$. We are ready for the definition.

Definition 5.3.1. Given sets A, B, their *product*, denoted $A \times B$, is the set

$$A \times B = \{(a, b) \mid a \in A \text{ and } b \in B\}.$$

There are two obvious *projection maps* $A \times B \to A$ and $A \times B \to B$, sending the pair (a, b) to a and to b respectively.

Example 5.3.2. In Example 5.2.1, (23) we presented the idea of a piece of furniture that was the same width as a space in the house. What if we say that ⌜a nice furniture placement⌝ is any space that is between 1 and 8 inches bigger than a piece of furniture? We can use a combination of products and pullbacks to create the appropriate type.



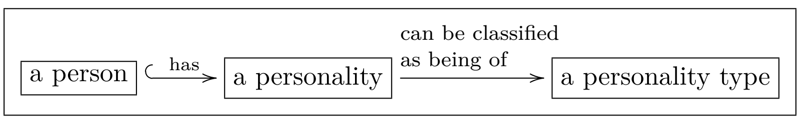
Here B and D are products and A is a pullback. This olog lays out what it means to be "a nice furniture placement" using products. The bottom horizontal aspect $B \to D$ is an example of a map obtained by the "universal property of products"; see Section 5.6.

5.3.3. *Products of more (or fewer) types.* The product of two sets A and B was defined in 5.3.1. One may also take the product of three sets A, B, C in a similar way, so the elements are triples (a, b, c) where $a \in A, b \in B$, and $c \in C$. In fact this idea holds for any number of sets. It even makes sense to take the product of one set (just A) or no sets! The product of one set is itself, and the product of no sets is the singleton set $\{*\}$. For more on this, see Section 5.5 or [Mac].

5.4. **Declaring an injective aspect.**

A function is called *injective* if different inputs always yield different outputs. For example the function that doubles every integer ($x \mapsto 2x$) is injective, whereas the function that squares every integer ($x \mapsto x^2$) is not because $3^2 = (-3)^2$. An example of an injective aspect is ⌜a woman⌝ $\xrightarrow{\text{is}}$ ⌜a person⌝ because different women are always different as people. An example of a non-injective aspect is ⌜a person⌝ $\xrightarrow{\text{has as father}}$ ⌜a person⌝ because different people may have the same father.

The easiest way to indicate that an aspect is injective is to use a "hook arrow" as in $f: A \hookrightarrow B$, instead of a regular arrow $f: A \to B$, to denote it. For example, the first map is injective (and specified as such with a hook-arrow), but the second is not in the olog:

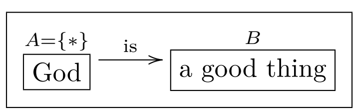


The author of this olog believes that no two people can have precisely the same personality (though they may have the same personality type).

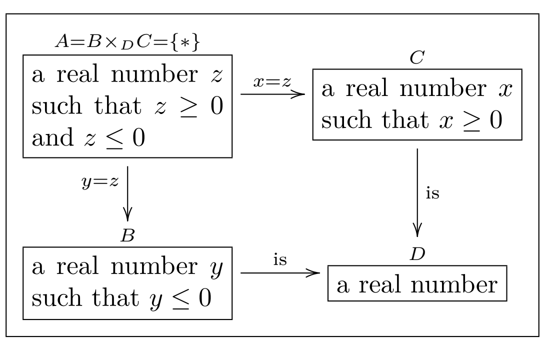
We include injective aspects in this section because it turns out that injectivity can also be specified by pullbacks. See [nL1] for details.

5.5. **Singletons types.**

A singleton set is a set with one element; it can be considered the "empty product." In other words if we denote $A^n = A \times A \times \cdots A$ (where A is written n times), then A^0 is the empty product and is a singleton set. One can specify that a certain type has only one instance by annotating it with $A = \{*\}$ in the olog. For example the olog

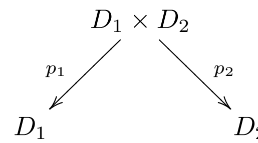


says that the author considers ⌜God⌝ to be single. As a more concrete example, the intersection of $\{x \in \mathbb{R} \mid x \geq 0\}$ and $\{y \in \mathbb{R} \mid x \leq 0\}$ is a singleton set, as expressed in the olog

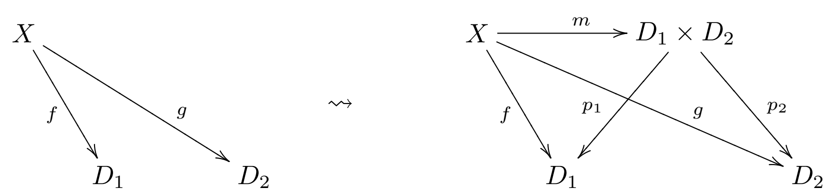


The fact that $A = B \times_D C$ and $A = \{*\}$ are declared indicates that there is only one possible instance of a real number that is in both B and C.

5.6. **The universal property of layouts.** I cannot do the notion of universal properties justice in this paper, but the basic idea is as follows. Suppose that $\mathcal{D}$ is an olog, that D_1, D_2 are types in it, and that $D = D_1 \times D_2$ (together with its projection maps $p_1 \colon D \to D_1$ and $p_2 \colon D \to D_2$) is their product.

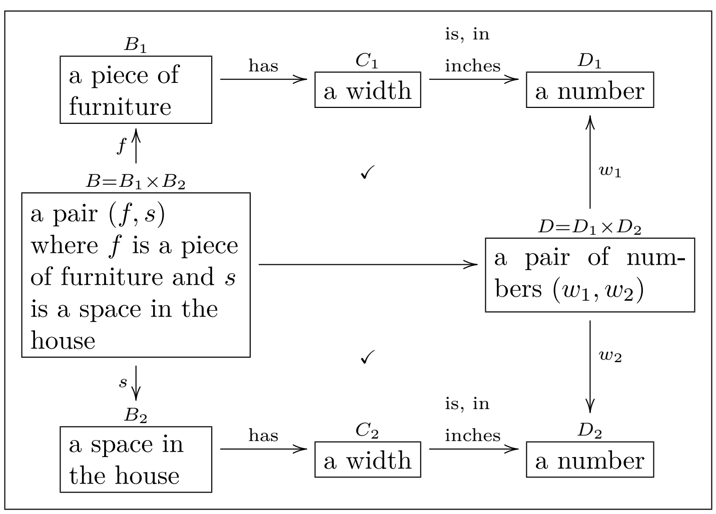
(24)


The so-called universal property of products should be thought of as "an existence and uniqueness" claim in $\mathcal{D}$. Namely, for any type X with maps $f \colon X \to D_1$ and $g \colon X \to D_2$, there is exactly one possible map $m \colon X \to D$ such that the facts $f = m; p_1$ and $g = m; p_2$ hold.

(25)


This may sound esoteric, but consider the following example.

The following olog is similar to the one in Example 5.3.2



Here the only unlabeled map is the horizontal one $B \to D$; how can we get away with leaving it unlabeled? How does a piece of furniture and a space in the house yield a pair of numbers? The answer is that B has a map to D_1 (the path across the top) and a map to D_2 (the path across the bottom), and hence the universal property of products gives a unique arrow $B \to D$ such that the two facts indicated by checkmarks hold. (In terms of (24) and (25) we are using $X = B$.) In other words, there is exactly one way to take a piece of furniture and a space in the house and yield a pair of numbers if we enforce that the first number is the width in inches of the piece of furniture and the second number is the width in inches of the space in the house.

At this point I hope it is clear that the universal property of products is a useful and constructive one. I will not describe the other universal properties (either for pullbacks, singletons, or any colimits); as mentioned above they can be found in [Awo].

6. More expressive ologs II

In this section I will describe various colimits, which are in some sense dual to limits. Whereas limits allow one to "lay out" a team consisting of many different interacting or non-interacting parts, colimits allow one to "group" different types together. For example, whereas the product of ⌜a number between 1 and 10⌝ and ⌜a letter between x and z⌝ has 30 elements (such as $(3, y)$), the coproduct of these two types has 13 elements (including 4). Just as "layout" is a too weak a word to capture the essence of limits, "grouping" is too weak a word to capture the essence of colimits, but it will have to do.

I will start by describing coproducts or "disjoint unions" in Section 6.1. Then I will describe pushouts in Section 6.2, wherein one can declare some elements in a union to be equivalent to others. There is a category-theoretic duality between coproducts and products and between pushouts and pullbacks. It extends to a duality between surjections and injections and a duality between empty types and singleton types, the subject of Sections 6.3 and 6.4. The interested reader can see [Awo] for details.

6.1. Coproducts.

Coproducts are also called "disjoint unions." If A and B are sets with no members in common, then the coproduct of A and B is their union. However, if they have elements in common, one must include both copies in $A \amalg B$ and differentiate between them. Here is a definition.

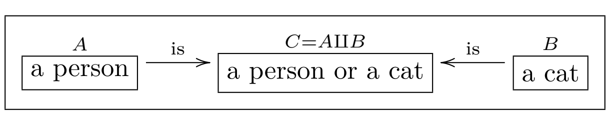
Definition 6.1.1. Given sets A and B, their *coproduct*, denoted $A \amalg B$, is the set

$$A \amalg B = \{(a, \text{“}A\text{”}) \mid a \in A\} \cup \{(b, \text{“}B\text{”}) \mid b \in B\}.$$

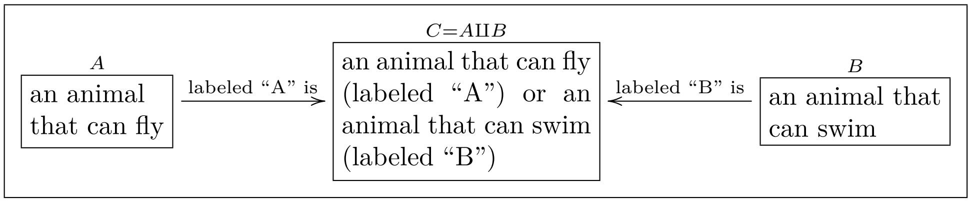
There are two obvious *inclusion maps* $A \to A \amalg B$ and $B \to A \amalg B$, sending a to $(a, \text{“}A\text{”})$ and b to $(b, \text{“}B\text{”})$, respectively.

If A and B have no elements in common, then the one can drop the "A" and "B" labels without changing the set $A \amalg B$ in a substantial way. Here are two examples that should make the coproduct idea clear.

Example 6.1.2. In the following olog the types A and B are disjoint, so the coproduct $C = A \amalg B$ is just the union.



Example 6.1.3. In the following olog, A and B are not disjoint, so care must be taken to differentiate common elements.



Since ducks can both swim and fly, each duck is found twice in C, once labeled as a flyer and once labeled as a swimmer. The types A and B are kept disjoint in C, which justifies the name "disjoint union."

6.2. **Pushouts.** Pushouts can express unions in which an overlap is declared. They can also express "quotients," where different objects can be declared equivalent. Given three objects and two arrows arranged as to the left, the pushout is drawn as the commutative square to the right:

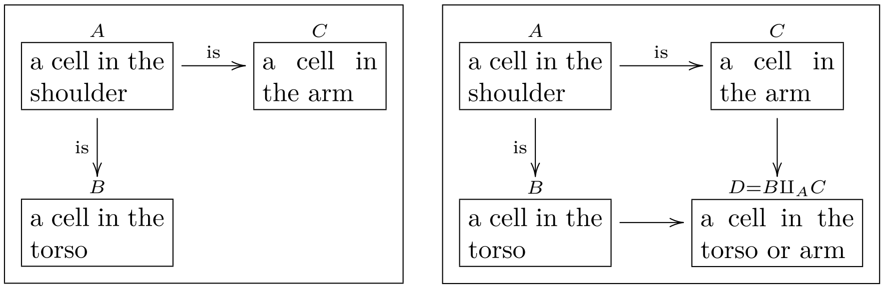
Given: $$\begin{array}{ccc} A & \xrightarrow{g} & C \\ {\scriptstyle f}\downarrow & & \\ B & & \end{array}$$ the pushout is drawn: $$\begin{array}{ccc} A & \xrightarrow{g} & C \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle f'} \\ B & \xrightarrow[g']{} & D. \end{array}$$

We write $D = B \amalg_A C$ and say "D is the pushout of B and C along A." The question is, what does it signify?

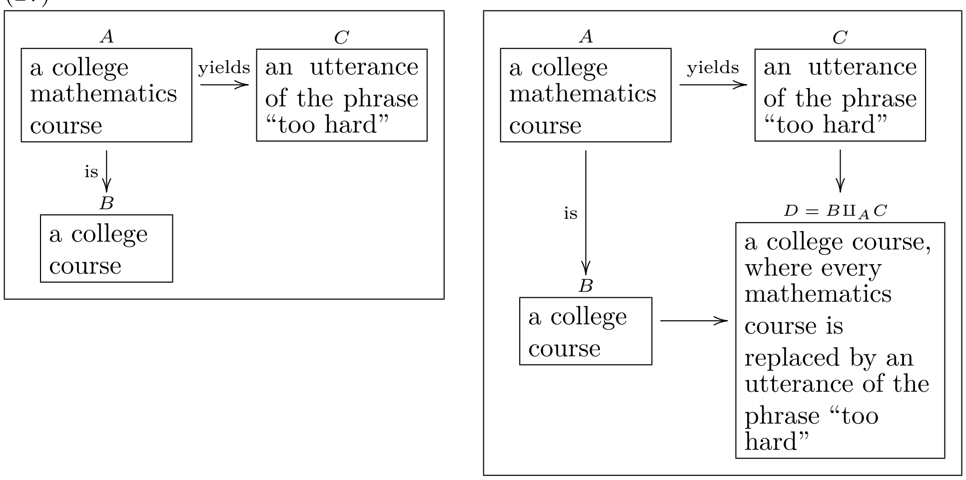
The idea is that an instance of the pushout $B \amalg_A C$ is any instance of B or any instance of C, but where some instances are considered equivalent to others. That is, for any instance of A, its B-aspect is considered the same as its C-aspect. This is formalized in Definition 6.2.2 after being exemplified in Example 6.2.1.

Example 6.2.1. In each example below, the diagram to the right is the pushout of the diagram to the left. The new object, D, is the union of B and C, but instances of A are equated to their B and C aspects. This will be discussed after the two diagrams.

(26)



(27)



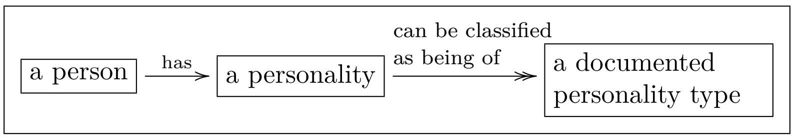
In Olog (26), the shoulder is seen as part of the arm and part of the torso. When taking the union of these two parts, we do not want to "double-count" the shoulder (as would be done in the coproduct $B \amalg C$, see Example 6.1.3). Thus we create a new type A for cells in the shoulder, which are considered the same whether viewed as cells in the arm or cells in the body. In general, if one wishes to take two things and glue them together, the glue serves as A and the two things serve as B and C, and the union (or grouping) is the pushout $B \amalg_A C$.

In Olog (27), if every mathematics course is simply "too hard," then when reading off a list of courses, each math course will not be read aloud but simply read as "too hard." To form D we begin by taking the union of B and C, and then we consider everything in A to be the same whether one looks at it as a course or as the phrase "too hard." The math courses are all blurred together as one thing. Thus we see that the power to equate different things can be exercised with pushouts.

Definition 6.2.2. Let A, B, and C be sets and let $f\colon A \to B$ and $g\colon A \to C$ be functions. The *pushout* of $B \xleftarrow{f} A \xrightarrow{g} C$, denoted $B \amalg_A C$, is the quotient of $B \amalg C$ (see Definition 6.1.1) by the equivalence relation generated by declaring $b \sim c$ (i.e. b is equivalent to c) if: $b \in B, c \in C$, and there exists $a \in A$ with $f(a) = b$ and $g(a) = c$.

6.3. **Declaring a surjective aspect.** A function $f\colon A \to B$ is called *surjective* if every value in B is the image of something in the domain A. For example, the function which subtracts 1 from every integer ($x \mapsto x - 1$) is surjective, because every integer has a successor; whereas the function that doubles every integer ($x \mapsto 2x$) is not surjective because odd numbers are not mapped to. The aspect is $\ulcorner$a published paper$\urcorner$ $\xrightarrow{\text{was published in}}$ $\ulcorner$an established journal$\urcorner$ is surjective because every established journal has had at least one paper published in it. The aspect is $\ulcorner$a published paper$\urcorner$ $\xrightarrow{\text{has as first author}}$ $\ulcorner$a person$\urcorner$ is not surjective because not every person is the first author of a published paper.

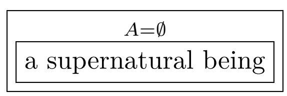
The easiest way to indicate that an aspect is surjective is to denote it with a "two-headed arrow" as in $f \colon A \twoheadrightarrow B$. For example, the second map is surjective (and indicated with a two-headed arrow) in the olog



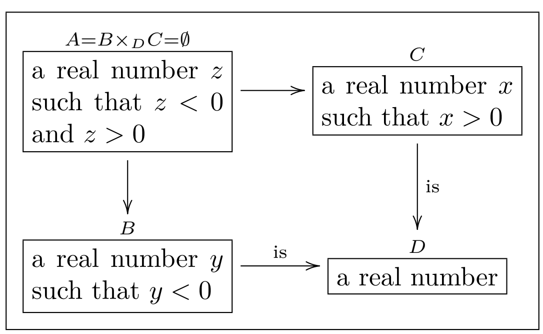
Here the first aspect is not considered surjective, presumably because the author imagines personalities had by no person.

We include surjective aspects in this section because it turns out that surjectivity can also be specified by pushouts. See [nL2] for details.

6.4. **Empty types.** The empty set is a set with no elements; it can be considered the "empty coproduct." In other words if we denote $n * A = A \amalg A \amalg \cdots \amalg A$ (where A is written n times), then $0 * A$ is the empty coproduct and is the empty set. One can declare a type to be empty by annotating it with $A = \emptyset$ in the olog. For example the olog



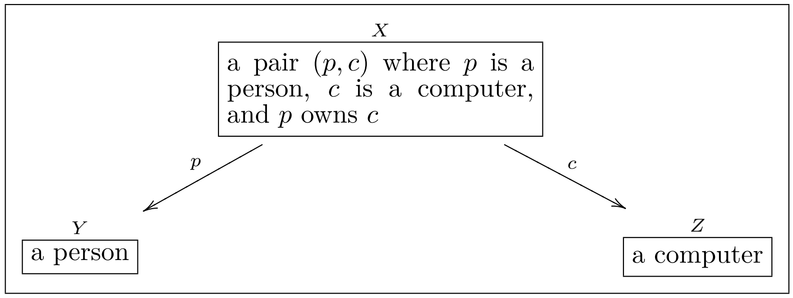
says that the set of supernatural beings is empty. As a more concrete example, the intersection of positive numbers and negative numbers is empty, as expressed in the olog



6.5. **Images.** In what remains of Section 6, I will discuss how the ideas of this section and the previous (Section 5) can be used together to create quite expressive ologs. First I will discuss how each aspect $f \colon A \to B$ has an "image," the subset of B that are "hit" by f. Then, in Sections 6.6 and 6.7, I will discuss how ologs can express all primitive recursive functions and many other mathematical concepts.

Consider the olog

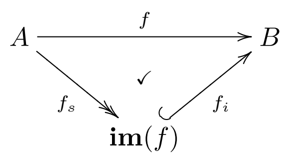
(28)



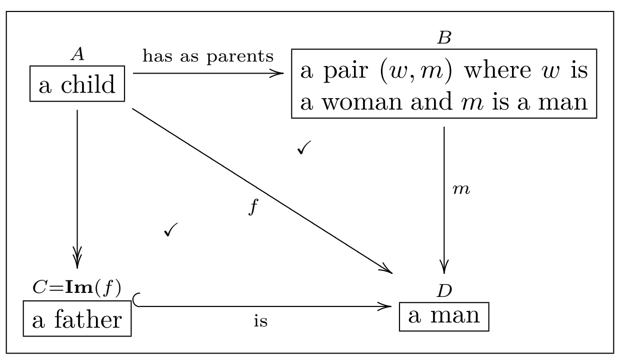
Some people own more than one computer, and some computers are owned by more than one person. Some computers are not owned by a person, and some people do not own a computer. The purpose of this section is to show how to use ologs to capture ideas such as "a person who owns a computer" and "a computer that is owned by a person". These are called the images of p and c respectively.

Every aspect has an image, and these are quite important for human understanding. For example the image of the map ⌜a person⌝ $\xrightarrow{\text{has as father}}$ ⌜a person⌝ is the type ⌜a father⌝. In other words, a father is defined to be a person x for which there is some other person y such that x is the father of y.

The image of a function $f\colon A \to B$ is a commutative diagram (fact)



where f_s is surjective and f_i is injective (see Sections 6.3 and 5.4). We indicate that a type is the image of a map f by annotating it with $\mathbf{Im}(f)$, as in the following olog:



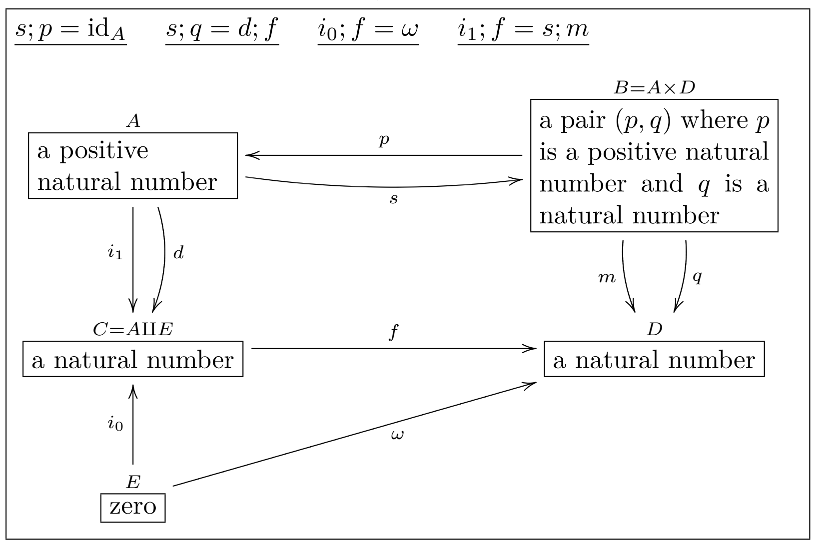
Hopefully it is also clear that ⌜a person who owns a computer⌝ and ⌜a computer that is owned by a person⌝ are the images of $p\colon X \to Y$ and $c\colon X \to Z$ (respectively) in Olog (28).

Using the label $\mathbf{Im}(f)$ is the easiest way to indicate an image, although one can also do so categorically using limits and colimits. See [Mac, Chapter VIII] for details.

6.6. **Application: Primitive recursion.** We have already seen how ologs can be used to express a conceptual understanding of a situation (all the ologs thus far exemplify this idea). In this section I hope to convince the reader that ologs are also able to express certain computations. In particular I will show by example that primitive recursive functions (like factorial or fibonacci) can be expressed by ologs. In this way, we can to tie computation and knowledge representation together within the same framework.

Example 6.6.1. In this example I will present an olog that can represent the "Factorial function," often denoted $n \mapsto n!$, where for example the factorial of 4 is 24. Recall that a *natural number* is any nonnegative whole number: $0, 1, 2, 3, 4, \ldots$.

$f(n) = n!$



The idea of this olog is to convey the factorial function as follows. A natural number is either zero or positive. Every positive natural number n has a decrement, $n-1$. The factorial of zero is 1. The factorial of a positive number n is obtained by multiplying n by the factorial of $n-1$.

To more explicitly describe the above olog, I must describe its intended instances. Hopefully the instances of each type (A through E) are self-explanatory, so I will describe the grouping, the layout, the aspects, and the facts. The set of natural numbers is the disjoint union of zero and the set of positive natural numbers and the maps i_0 and i_1 are the inclusions into the coproduct, which explains the grouping $C = A \amalg E$. The layout $B = A \times D$ is self-explanatory, and the maps p and q are the projections from the product. The map d is the decrement map $n \mapsto n-1$, the map ω sends 0 to 1, the map m is multiplication $(n, n') \mapsto n * n'$. Once m, d, and ω are so-defined, the first two facts ($s;p = \mathrm{id}_A$ and $s;q = d;f$) specify that s sends n to the pair $(n, f(d(n)))$, and the second two facts specify that f sends 0 to 1 and sends a positive number n to $m(s(n)) = m(n, f(d(n)))$, i.e. n goes to the product $n * (n-1)!$.

The above olog defines the factorial function (f) in terms of itself, which is the hallmark of primitive recursion. Note, however, that this same olog can compute many things besides the factorial function. That is, nothing about the olog says that the instances of ⌜Zero⌝ is the set $\{0\}$, that ω sends 0 to 1, that d is the decrement function, or that m is multiplication — changing any of these will change f as a function. For example, the same olog can be used to compute "triangle numbers" (e.g. f(4)=1+2+3+4=10) by simply changing the instances of ω and m in the obvious ways (use $\omega = 0, m = +$ rather than $\omega = 1, m = *$)). For a radical departure, fix any forest (set of graphical trees) F, let $E =$ ⌜zero⌝ represent its set of roots, A the other nodes, ω the constant 0 function, d the parent function, and m sending $(p, d(p))$ to $f(d(p))+1$. Then for each tree in F and each node n in that tree, the function f will send n to its height on the tree.

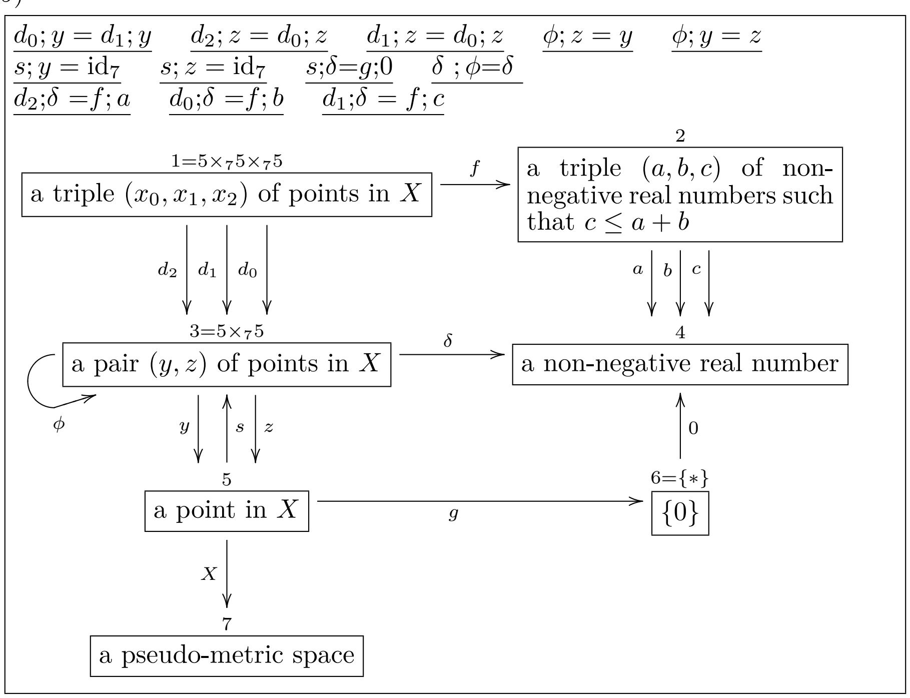
Primitive recursion is a powerful technique for deriving new functions from the repetition of others using a kind of "while loop." The general form of primitive recursive functions can be found in [BBJ], and it is not hard to imitate Example 6.6.1 for the general case.

6.7. **Application: defining mathematical concepts.** In this subsection I hope to convince the reader that many mathematical concepts can be defined by ologs. For example, I will recall the definition of pseudo-metric space (in 6.7.1) and then provide an olog with the same content (in 29).

Definition 6.7.1. Let $\mathbb{R}_{\geq 0}$ denote the set of non-negative real numbers. A *pseudo-metric space* is a pair (X, δ) where X is a set and $\delta\colon X \times X \to \mathbb{R}_{\geq 0}$ is a function with the following properties for all elements $x, y, z \in X$:

(1) $\delta(x, x) = 0$;
(2) $\delta(x, y) = \delta(y, x)$; and
(3) $\delta(x, z) \leq \delta(x, y) + \delta(y, z)$.

(29)



As long as the instances for the right-hand side of this olog are mathematically correct (i.e. we assign 4 the set of non-negative real numbers), this olog has the same content as Definition 6.7.1. One can use ologs to define usual metric spaces (in which Property (1) in Definition 6.7.1 is strengthened), but it would have taken too much space here. This brings us to a clear disadvantage of using ologs: they are bigger and harder to read – ologs can get "clunky". However, they provide a more precise description, relying less on the grammar of English and more on the mathematical "grammar" of sets and functions. Thus an olog is more likely to be instantly readable by a theorem prover such as Coq ([Coq]). Moreover, various parts of this olog may be reusable in other contexts, and hence connect pseudo-metric spaces into a web of neighboring definitions and theorems.

In fact, once a corpus of mathematics has been written in olog form, evidence of conjectures not yet proven could be written down as instance data. For example, one could record every known prime as instances of a type ⌜prime⌝ and a machine could automatically check that Goldbach's conjecture (written as an olog containing ⌜prime⌝ as a type) holds for all example "so far." With definitions, theorems, and examples all written in the same computer-readable language of ologs, one may hope for much more advanced searching and knowledge retrieval by humans. For example, one could formulate very precise questions as database queries and use SQL on the database corresponding to a given olog (see Section 3.2).

7. Further directions

Ologs are basically categories which have text labels to explain their intended semantic. As such there are many directions to explore ranging from quite theoretical to quite practical. Here we consider three main classes: extending the theory of ologs, studying communication with ologs, and implementing ologs in the real world.

7.1. **Extending the theory of ologs.**

In this paper I began by discussing basic ologs, which are rich enough to capture the semantic of many situations. In Sections 5 and 6 I added more expressivity to ologs to allow one to encode ideas such as intersections, unions, and images. However, ologs could be even more expressive. One could add "function types" (also known as exponentials); add a "subobject classifier type," which could allow for negation and complements as well as power-sets; or even add fixed sets (like the set of Strings) to the language of ologs. This is not too hard (using sketches, see [Mak]); the reason I did not include them in this paper was more because of space than any other reason.

Another generalization would be to allow the instances of an olog to take values in a category other than **Set**. For example, one could have an instance-space rather than an instance-set, e.g. it is clear that the instances of the type ⌜a point on the unit circle⌝ constitute a topological space. One could similarly argue that the instances of the type ⌜a human invention⌝ have a topology or metric as well (e.g. as an invention, the cellphone is closer to the telephone than it is to artificial flavoring). Instance data on an olog $\mathcal{C}$ corresponds to a functor $\mathcal{C} \to \mathbf{Set}$ in this paper, but it is quite easy to replace **Set** with a different category such as **Top** (the category of topological spaces), and this may have interesting uses in data modeling.

In Section 6.7, I explicitly showed that pseudo-metric spaces (and I stated further that metric spaces) can be presented by ologs. It would be interesting to see if theorems could also be proven entirely within the context of ologs. If so, a teacher could first sketch a mathematical proof as a small or sparse olog $\mathcal{C}$, and then use a functor $\mathcal{C} \to \mathcal{D}$ to rigorously "zoom in" on that proof so that the sketch becomes a full-fledged proof (as the maps in $\mathcal{C}$ are factored into understandable units in $\mathcal{D}$).

If ologs are to be viable venues in which to discuss results in mathematics, then they should be capable of describing all recursion, not just primitive recursion (as in Section 6.6). I do not yet have an understanding for how this can be done. If recursion can be fully defined with the ologs described above, it would be interesting to see it written out; if not, it would be interesting to understand what basic idea could be gracefully added to ologs so that recursion becomes expressible.

In a different direction, one could test the expressive power of ologs by defining simple games, like Tic Tac Toe or Chess, using ologs. It would be impressive to define a vocabulary for writing games and a program which could automatically convert an olog-defined game into a playable computer game. This would show that the same theory that we have seen express ideas about fatherhood and factorials can also be used to invent games and program computers.

7.2. **Studying communication with ologs.** As discussed in Section 4, ologs can be connected by functors into networks that are not just 2-way, but n-way. These communication networks should be studied: what kinds of information can pass, how reliable is it, how quickly can it spread, etc. This may be applicable in fields from economics to psychology to sociology. Such research may use results from established mathematics such as Network Coding Theory (see [YLC]).

In [SA], we study how two or more entities (described as ologs) can communicate new ideas (not just new instance data) to each other. It would be interesting to see how well this "communication protocol" works in practice, and whether it can be theoretically automated. Furthermore, this communication protocol and any theoretical automation of it should be implemented on a computer to see if different database schemas can be meaningfully integrated with minimal human assistance.

It may be possible to train children to create ologs about their interests or about a given lesson. These ologs would show how the child actually perceives something, which would probably be fascinating. By my experience and that of people I have taught, the process of building an olog usually leads to a clarification of the concepts involved. Moreover, a class project to connect the ologs of different students and between the students and the teacher, may have excellent pedagogical benefits.

Finally, it may be interesting to study "local truth" vs. "global truth" in a network of ologs. Functorial connections between ologs can allow for translation of ideas between members of a group, but there may be ideas which do not extend globally, just as a Möbius band does not admit a global orientation. That is, given three parties on the Möbius band, any pair can agree on a compass orientation, but there is no choice that the three can simultaneously agree on. Similarly, whether or not it is possible to construct a global language which extends all the existing local ones could be determined if these local languages and their connections were entered into a computer olog system.

7.3. **Implementing ologs in the real world.** Once ologs are implemented on computers, and once people learn how to author good ologs, much is possible. One advantage comes in searching the information space. Currently when we search for a concept (say in Google or on our hard drive), we can only describe the concept in words and hope that those words are found in a document describing the concept. That is, search is always text-based. Better would be if the concept is meaningfully interconnected in a web of concepts (an olog) that could be navigated in a meaningful (as opposed to text-based) way.

Indeed, this is the semantic web vision: When internet data is machine-readable, search becomes much more powerful. Currently, we rely on RDF scrapers that scour web pages for ⟨subject, predicate, object⟩ sentences and store them in RDF format, as though each such sentence is a fact. Since people are inputting their data as prose text, this may be the best available method for now; however, it is quite inaccurate (e.g. often 15% of the facts are wrong, a number which can lead to degeneration of deductive reasoning – see [MBCH]). If ideas could be put on the internet such that they compatibly made sense to both human and computer, it would give a huge boost to the semantic web. I believe that ologs can serve as such a human-computer interface.

While it is often assumed that because we all speak the same language we all must mean the same things by it, this is simply not true. The age-old question about whether "blue for me" is the same as "blue for you" is applicable to every single word and idiom in our language. There is no easy way to sync up different people's perceptions. If communication is to be efficient, agreements must be fairly explicit and precise, and this precision demands a rigor that is simply unavailable in English prose. It is available in a network of ologs (as described in Section 4).

For example, the laws of the United States are hopelessly complex. Residents of the US are required to obey the laws. However, unlike the rules of the Scholastic Aptitude Test (SAT), which take 10 minutes for the proctor to read aloud, the laws of the US are never really expressed — the most important among them are hopefully picked up by cultural osmosis. If an olog was created which had enough detail that laws could be written in that format, then a woman could research for herself whether her landlord was required to fix her refrigerator or whether this was her responsibility. It may prove that the olog of laws is internally inconsistent, i.e. that it is impossible for a person to satisfy all the laws — such an analysis, if performed, could fundamentally change our outlook on the legal system.

The same goes for science; information written up in articles is much less accessible than information that is entered into an ontology. However, the dream of a single universal ontology is untenable ([Min]). Instead we must allow each lab or institute to create its own ontology, and then require citations to be functorial olog connections, rather than mere silo-to-silo pointers. Thus, a network of ologs should be created to represent the understanding of the modern scientific community as a multi-faceted whole.

Another impetus for a scientist to write an olog about the study at hand is that, once an olog is made, it can be instantly converted to a database schema which the scientist can use to input all the data pertaining to this study. Indeed, if some data did not fit within this schema, then the olog must have been insufficient to begin with and should be modified to fully describe the experiment. If scientists work this way, then the separation between them and database modelers can be reduced or eliminated (the scientist assumes the database modeling role with little additional burden). Moreover, if functorial connections are established between the ologs of different labs, then data can be meaningfully shared along those connections, and ideas written in the language of one lab's olog can be translated automatically into the language of the other's. The speed and accuracy of scientific research should improve.

References

[Awo] S. Awodey. *Category theory.* Second edition. Oxford Logic Guides, 52. Oxford University Press, Oxford, 2010.

[BBJ] G.S. Boolos, J.P. Burgess, R.C. Jeffrey. *Computability and logic.* Fifth edition. Cambridge University Press, Cambridge, 2007.

[BW1] M. Barr, C. Wells. *Category theory for computing science.* Prentice Hall International Series in Computer Science. Prentice Hall International, New York, 1990.

[BW2] M. Barr, C. Wells. *Toposes, triples and theories.* Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences], 278. Springer-Verlag, New York, 1985.

[Bor] A. Borgida. "Knowledge Representation meets Databases Ñ a view of the symbiosis Ñ" 20th International Workshop on Description Logics, 2007.

[CM] M. Chein, M-L Mugnier. "Graph-based Knowledge Representation and Reasoning — Computational Foundations of Conceptual Graphs." Advanced Information and Knowledge Processing Series, Springer London, 427 pages, 2008.

[Coq] *The Coq proof assistant*, 2011. http://coq.inria.fr/.

[HC] M.J. Healy and T.P. Cavdell. "Neural Networks, Knowledge and Cognition: A Mathematical Semantic Model Based upon Category Theory," UNM Technical Report EECE-TR-04-020, DSpaceUNM, University of New Mexico. 2004

[LS] F.W. Lawvere, S.H. Schanuel. *Conceptual mathematics. A first introduction to categories.* Second edition. Cambridge University Press, Cambridge, 2009.

[Mac] S. Mac Lane. *Categories for the working mathematician.* Second edition. Graduate Texts in Mathematics, 5. Springer-Verlag, New York, 1998.

[Mak] M. Makkai. "Generalized sketches as a framework for completeness theorems. I,II,III." J. Pure Appl. Algebra 115 (1997), no. 1.

[Min] G.W. Mineau. "Sharing knowledge: Starting with the integration of vocabularies." *Lecture Notes in Computer Science*, Vol 754 (1993), pp. 34-45. Springer.

[MBCH] T.M. Mitchell, J. Betteridge, A. Carlson, E. Hruschka. "Populating the semantic web by macro-reading internet text." *Lecture Notes in Computer Science*, Vol. 5823 (2009), pp. 998-1002. Springer.

[nL1] nLab contributors. "Monomorphism." nLab. http://ncatlab.org/nlab/show/monomorphism

[nL2] nLab contributors. "Epimorphism." nLab. http://ncatlab.org/nlab/show/epimorphism

[Pie] B.C. Pierce. *Basic Category Theory for Computer Scientists.* MIT Press, 1991.

[Sic] G. Sica. "What is category theory?" 2006, Polimetrica S.a.s. Milan, Italy.

[Sow] J. Sowa. "Semantic Networks." Available online: http://www.jfsowa.com/pubs/semnet.htm

[Sp1] D.I. Spivak. "Higher dimensional models of networks," 2009. Available online: http://arxiv.org/pdf/0909.4314

[Sp2] D.I. Spivak. "Functorial data migration," 2010. Available online: http://arxiv.org/abs/1009.1166

[Sp3] D.I. Spivak. "Categories via graphs and paths," 2011. To appear online: http://math.mit.edu/∼dspivak/cs/cats.pdf

[SA] D.I. Spivak, M. Anel. "Communication protocol." In preparation.

[W1] Wikipedia contributors. "Amino acid." Wikipedia, The Free Encyclopedia. 30 Sep. 2010. Web. 30 Sep. 2010. http://en.wikipedia.org/wiki/Amino_acid

[YLC] R. Yeung, S-Y Li, N. Cai. "Network coding theory," *Foundations and trends in communications and information theory.* now Publishers Inc., Boston 2006.

Department of Mathematics, Massachusetts Institute of Technology, Cambridge MA 02139

E-mail address: dspivak@math.mit.edu