3-22-2011

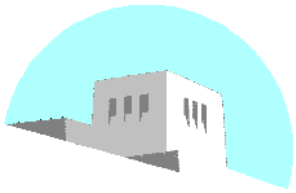
A Categorical Model for Faceted Ontologies with Data Repositories

Michael Healy

Renzo Sanchez-Silva

Thomas Caudell

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING



SCHOOL OF ENGINEERING UNIVERSITY OF NEW MEXICO

A Categorical Model for Faceted Ontologies with Data Repositories[1]

Michael J. Healy
Department of Electrical and Computer Engineering
e-mail: mjhealy@ece.unm.edu

Renzo C. Sanchez-Silva
Department of Electrical and Computer Engineering
e-mail: renzo@ece.unm.edu

Thomas P. Caudell
Department of Electrical and Computer Engineering and
Department of Computer Science
e-mail: tpc@ece.unm.edu

University of New Mexico
Albuquerque, New Mexico 87131, USA

UNM Technical Report: EECE-TR-11-0002

Report Date: March 22, 2011

[1]This work was supported by the Defense Threat Reduction Agency (DTRA), United States Department of Defense, under Contract No. DTRA01-03-D-0009.

Abstract

We present a theoretical framework for faceted ontologies equipped with data repositiories. The ontology structures, inferencing within ontologies, and repository use and updating are expressed using mathematical structures given by category theory.

Keywords

Ontology, Facet, Category Theory

1 Introduction

The objective of our work is a methodology for the development, maintenance, and application of faceted ontologies equipped with data repositories. This report presents an initial theoretical approach for faceted ontologies with repositories.

An ontology is an expression of that which exists, that is, objects, properties, events, processes and their relationships [18] in a universe of discourse, which we shall call a world. A faceted ontology is a sort of "ontology with faces", each face an ontology specialized to a particular viewpoint on the world. Any investigation of ontologies and the items they are supposed to express necessarily involves the symbolic representation of knowledge about things and their relationships in some conception of a world. In the recent application of ontology outside philosophy, it also involves the correspondence of the symbolic representation to data, which we consider in the form of a mathematical model of, say, a computer system such as an online data repository [15]. Coupling an ontology to data gathered from the world it is supposed to describe requires that it have an unambiguous semantics, the meaning of its symbolic structures as determined by a systematic interpretation of them in the world environment. In the work presented here, this requirement is addressed with mathematical rigor, the idea being to disambiguate the semantics of symbolic representations so that their correspondence with data is accurate and precise. This will facilitate the development of computerized systems that allow the exploitation of ontologies, for example in performing updates to relational data repositories and in gaining useful information from the data. To this end, we introduce a theory of faceted ontologies based upon category theory, the mathematical theory of structure (see the Appendix for a brief introduction). In this investigation there are three kinds of structure.

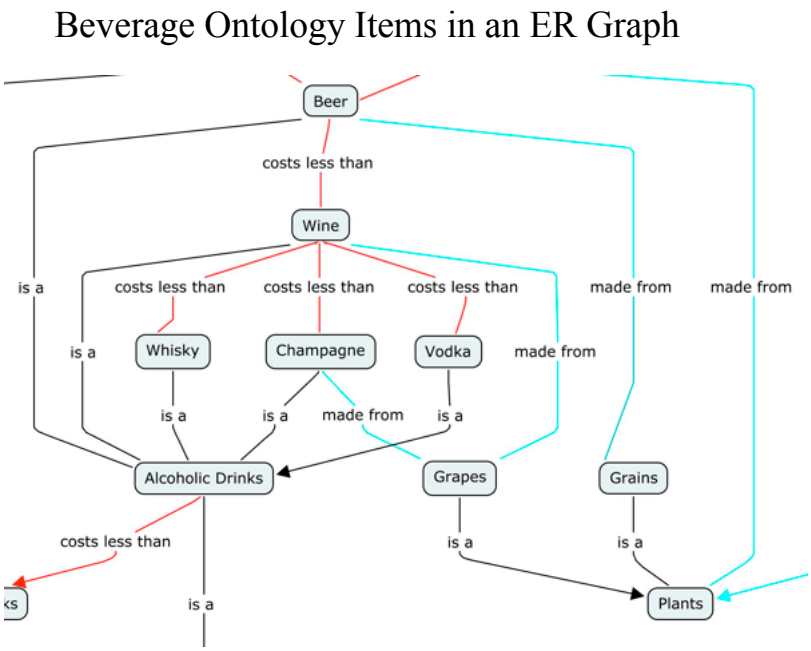


Figure 1: **The Beverage Ontology as an entity-relationship (ER) graph.**

The first kind of structure is that of an ontology. This is expressed as a network of concepts, which are human-understandable descriptions of different kinds or classes of things, and descriptive links showing how the things of one concept relate to those of another. In an ontology for beverages, part of which is shown in the form of an entity-relationship (ER) graph in Figure 1, the entities `Beer`, `Wine`, etc. have `is_a` links to the entity `Alcoholic`

Drinks. The entities Grapes and Grains have is_a links to Plants. There is also a made_from link from Wine to Grapes and another from Beer to Grains. The terminology is suggestive: A link labelled "A is_a B" is meant to express a subclass relationship, as in "every member of the class of A's is a member of the class of B's". The class of beers, whose members are instances of the concept Beers, is a subclass of the class of instances of the concept Alcoholic Drinks. Of course, this is meant to express the common understanding that every beer is an alcoholic drink. It is tempting to say that the meaning of the graph with its labelled nodes and links is clear. But this very thought suggests its opposite, that the meaning is not so clear given that our objective is to apply ontology to computer systems.

The notions embodied in the graph are clear only to we humans, and only because the entities and the relationship links are labelled with familiar expressions derived from natural language such as Beer, is_a, Grains and made_from. Labels alone are insufficient for a computer system for ontology and data manipulation because it must be programmed specifically to enforce the understanding implicit in the labels. For example, in symbolic inferencing with the ontology, the computer system can find the entities that are subconcepts of Alcoholic Drinks only if it is programmed for symbolic manipulation and can recognize an "is_a" link and associate the symbolic expression with its meaning in terms of the search operation to be performed. The programming involved must convey the semantics implied by the graph, that is, the intended meaning of the structure of labelled entities and links. This requires a translation of symbol system into computer system with a human-friendly interface. The translation requires some degree of formalization, preferably made explicit in a system specification or at least in a thorough system document once developed. Ontology development systems such as OWL are programmed to recognize properly-formed symbolic expressions and enforce the labelled graph semantics as indicated. The online OWL documentation [16] contains a discussion of the issue of translation to computer systems in the context of the World Wide Web. The relationship between the semantics of OWL and our mathematical formalism will be discussed in Section 2.

The second kind of structure is that of a relational data repository associated with an ontology. A repository contains the data for the classes whose concepts are in the ontology, together with mapping links, or correspondences, which link the data items of one class to those of another. As shown in Figure 2, there is a mapping link between data classes in the repository for each symbolic link between concepts in the ontology. A relational database is one example of this kind of structure, where the data items for a concept form a collection that we shall think of as a finite set. A mapping link then corresponds (by restructuring the graph, if necessary) to a function that maps the elements of one set to elements in the other. Another way of representing a collection is as a column in a table, and the mapping links as rows. Here again the issue of semantics arises if the repository is to ensure the integrity of its data: the concepts and their links must be matched by the appropriate sets of data and mapping links. Without attention to this, data for different concepts might appear in the same place (such as a column), and the mappings associated with the links might be incorrect. Queries would then yield unexpected and incorrect results.

The third kind of structure is suggested by the fact that the classes and relational links of a repository are associated with an ontology. This association is a special kind of mapping or correspondence (the large arrow between the ontology and repository in Figure 2) which associates each concept of the ontology with its associated data items (a set, or a column in a table) and each link of the ontology with a data mapping (one of the small arrows, a function or a row in a table). As will be seen, these three structures require more information than a graph conveys to properly express the semantics of an ontology and any data associated with it. For now, the labelled graph and the other descriptions of structure will suffice; the analysis acquires the needed depth in Section 2.

As discussed earlier, the Beverage Ontology concerns different beverages such as wine and beer and also their principle ingredients such as grapes and grains. Suppose that this ontology has users who are specialists of different kinds; one user, such as a brewmaster, might be concerned with how the beverages are made (mixing, fermentation, brewing, etc.), another with marketing and pricing information (sold by the bottle, price varying within such-and-such a range), and another specifying their recommended use (for example, whether for social drinking or to accompany dining, and with which foods if for dining). An all-inclusive ontology is desirable to maintain the organization of all the information present in the graph of Figure 1. The different kinds of

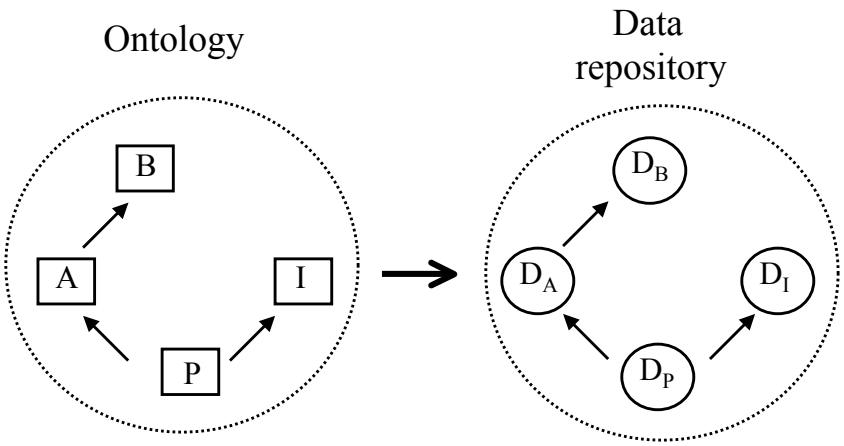


Figure 2: **Three structures: an ontology, a data repository, and a correspondence between the two.**

users, however, might find the inclusion of all the information associated with each concept at best cumbersome and at worst confusing. Faceted ontologies are meant to address this by providing additional structures which correspond to the concepts and links of the all-inclusive ontology but provide a different interface or *facet* for each type of specialist. The facets access the information specific to their specialties through the all-inclusive or main ontology. In turn, the main ontology maintains the coherence of the total body of information present; it associates the information for each concept specific to one facet with that for the same or a similar concept in another facet, and associates the links likewise.

The information in each facet can be expressed in terminology understandable by its type of specialist, while the main ontology maintains the information in a "neutral" language. This is indicated in Figure 3, where the legend "x" with arrows to "a" and "b" inside the circle for Main Ontology indicates that it maintains information for specialties "a" and "b" combined and that this is translated into the specialty languages. Facet 1, on the other hand, is shown as having a node-and-link structure structure identical to that of the main ontology but containing only information for specialty "a". If the Main Ontology contains information for how beverages are made, how they are priced and marketed, and their intended use, then Facet 1 contains only the information on how they are made, Facet 2 contains only the information on pricing and marketing, and so forth. The node-and-link structure of facets 1 and 2 may duplicate that of the main ontology, but only facet-specific information is labelled. For example, if Facet 2 is concerned only with pricing and marketing of the beverages themselves and not at all with their ingredients such as grapes, then the concept `Wine` is shown but the concept `Grapes` and the `is_a` link from `Wine` to `Grapes` need not be labelled as such except in the main ontology. For flexibility, however, it is desirable to include them in some form unobtrusive to the Facet 2 user. For example, at a future time it might become desirable for Facet 2 to include information about the pricing of grapes, since that can contribute to the price of wine. Hence, the ability to have the entire structure but strongly highlight and label only the relevant concepts and links can be useful. Another consideration is the ability to perform data repository updates through the facet interface that are *propagatable*. As will be seen, an update to one class in the repository can require changes in its links to other classes. This requires that the facet user have access to the salient links.

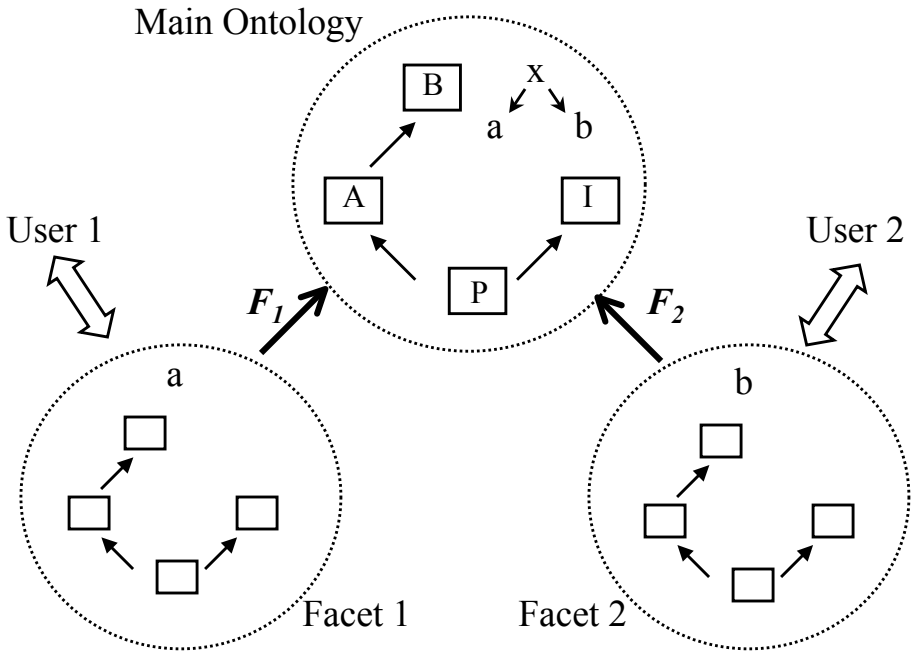


Figure 3: **A faceted ontology.**

Faceted ontologies yield additional examples of the third kind of structure, the correspondence between structures of the first and second kinds, except that now the correspondence is between ontologies. For example, the arrows labelled F_1 and F_2 in Figure 3 are correspondences between the main ontology and Facets 1 and 2. These correspondences are programmed in the software system that maintains the faceted ontology. The users interacting with their facets can make facet-specific queries which are forwarded to the main ontology. This in turn resolves the query and supplies this information to the facet for access by the user. In this way, the main ontology can be linked through a correspondence mapping to the data repository and the requested data retrieved via F_1, say, in a facet-specific format. Facet-specific inferencing can be performed in a similar manner, through an inference engine—a symbolic manipulation system—that operates on Main Ontology.

The remaining sections of this report present a mathematical semantic approach to faceted ontologies. The idea here is to resolve the issues raised about an ontology and its meaning in terms of operations on and with the ontology in inferencing, data manipulation, querying, and so forth. In Section II, we discuss ontologies and repositories as mathematical categories and correspondence mappings called functors and natural transformations. In Section III, we describe facets based upon the mathematical theory presented in Section II. Section IV concerns the repository modeling based upon the foregoing, and sections IV and V show how category theory is applied to ensure propagatable updates.

2 The categorical approach

Our approach to faceted ontologies is based on [9], where category theory is applied to formalize data base schemas by regarding them as theories in formal logic. The Beverage Ontology entity-relationship graph of Figure 1 is like a schema graph and bears some resemblance to the graphical depiction of a theory. However, interpreting the labels for the nodes and links of an ER graph requires human intuition. A formalization, on

the other hand, expresses enough of the intended semantics of the ontology to allow mechanical inferencing and manipulation of data in a manner consistent with it. Here, we obtain a formalization by completing the graph to form a category **Bev**, called the classifying category of the theory. Since category theory is the mathematical formalism used here but is unfamiliar to many, the Appendix contains a brief tutorial. In this section, we assume a sufficient background and explain how the ontology formalization works.

2.1 Categorical completion

Initially in forming the categorical completion, the objects of the category are the concepts, or entities, at the nodes of the graph of Figure 1 (but many new ones will be added, as will be explained). The morphisms, or arrows, of the category begin with the links, or relationships, of the graph. More objects and arrows are obtained recursively as specified in the following discussion to obtain the categorical completion. In a category, for any two arrows $f\colon a \longrightarrow b$ and $g\colon b \longrightarrow c$, there is an arrow $g \circ f\colon a \longrightarrow c$ which can be obtained as the composition of f and g. If the latter two arrows are links in a graph, we might obtain $g \circ f$ simply by concatenating them, since they form a path through b. A meaning is attached to $g \circ f$ based upon the meanings of its factors a and b. For example, the link `made_from`: `Wine` $\longrightarrow$ `Grapes` clearly (to us humans, because of the use of language) means that wine is made from grapes, and the link `is_a`: `Grapes` $\longrightarrow$ `Plants` means that grapes are plants. Given that those meanings have been established, and regarding the links as arrows in a category, it is easy for us to assign the meaning "wine is made from plants" to the composition `is_a` $\circ$ `made_from`: `Wine` $\longrightarrow$ `Plants` as shown in Figure 4. In fact, this use of composition has produced a deduction, a form of inference: wine is made from grapes, grapes are made from plants; therefore, wine is made from plants. But as mentioned in the Introduction, the semantics of the names assigned to the objects and morphisms must be specifically implemented in a computer system for working with the ontology if the composition is to correctly convey the meaning.

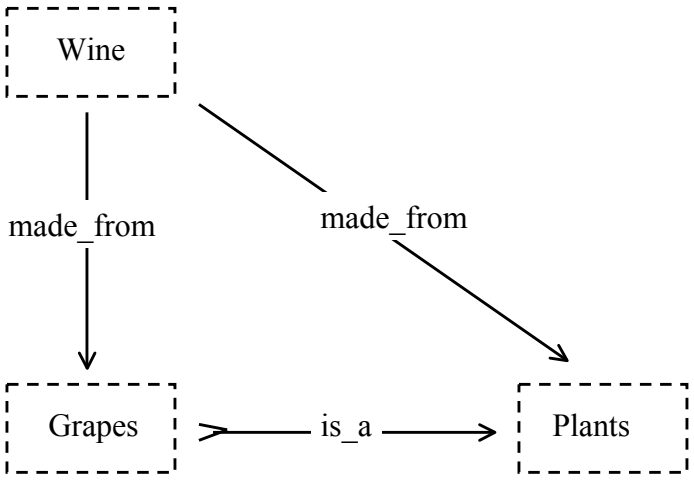


Figure 4: **A composition relating two** `made_from` **arrows.**

The categorical formalization may well indicate links that were missing in the original graph. Notice that

the entities `Beer`, `Wine`, etc. have `is_a` links to the entity `Alcoholic Drinks`. The entities `Grapes` and `Grains` have `is_a` links to `Plants`. There is then a `made_from` link from `Wine` to `Grapes`, another from `Beer` to `Grains`, but no `made_from` link from `Alcoholic Drinks` to `Plants`. Completeness would seem to call for such a link, and to require that it be consistent with the other links mentioned. If beer, wine, whisky, champagne, vodka are alcoholic drinks, and are the only ones represented in the ontology, and all are made from grapes or grains, which are plants, then why can not we conclude that alcoholic drinks are made from plants? Recall that another type of missing link, the compositions of arrows, have been included by forming the categorical completion of the graph. In subsection 2.2, we shall enlarge upon the categorical completion to form the classifying category of a theory, which is how we formalize an ontology. In so doing, we include the missing link from `Alcoholic Drinks` to `Plants`, thereby making direct use of the composition of arrows. But of most significance are other categorical structures added to the completion. Before discussing these structures in detail, we provide an overview of their use (refer to the Appendix for definitions).

Additional constraints can be imposed by enlarging upon the categorical completion. First, the category is augmented with a terminal object and all the arrows having it as codomain. Second, certain diagrams in the initial completion can be declared to commute; this is accomplished by specifying that certain pairs of paths from the augmented graph yield the same composition arrow. Third, and this step is quite extensive, we can decree that all finite coproducts and all pullbacks exist. This leads to the inclusion of additional objects and arrows, for each coproduct has arrows with domains consisting of all the objects in a finite discrete diagram and a common codomain consisting of an apical object—the coproduct. Each pullback has arrows with codomains consisting of all the objects in a diagram containing two arrows with a common codomain, and which share a domain consisting of an apical object—the pullback. By specifying that the category include coproducts for all finite, discrete diagrams and pullbacks for all appropriate diagrams, we obtain **Bev** as an infinite category. Some of the coproducts and pullbacks can be specified to specifically include objects and morphisms derived directly from the ER graph. The greater share of the finite coproducts and finite pullbacks can be calculated incrementally based on a theorem in category theory. This begins with diagrams formed from the links and compositional paths of the ER graph augmented with a terminal object and accompanied by the initial, specified set of commutative diagrams. The resulting cones and cocones can be used in further diagrams to form yet more coproducts and pullbacks, and so forth in a never-ending multistage process. In practice, only those items needed can be calculated "on-the-fly". Also , because of the theorem mentioned, the inclusion of a terminal object in the category enables us to obtain *all* finite limits—that is, limits for all finite diagrams. Thus, with the addition of any missing links as discussed in the previous paragraph, **Bev** becomes the classifying category of a theory with a considerable expressive power.

It was stated that the ER graph of Figure 1 did not fully express the meaning implied by the labels used for entites and relationships. Now, however, with the classifying category **Bev** of a full theory in hand, we have access to a formalization of the intended semantics. The objects of the category **Bev** such as `Wine` and `Alcoholic Drinks` are types, representing the classes of items the ontology or theory is about. We call these *sorts*, and our theory is called a sorted theory (predicates can be used to correspond to the objects of the category in place of sorts to obtain an unsorted theory). Each arrow of the category is to be interpeted as an operation on one sort—its domain, intepreted as a class of items—that maps its members to members of the class represented by its codomain sort. Because there are different links with the same label inherited from the graphs, we need to distinguish mathematically between them as arrows; hence, we use the abbreviated terms $\texttt{made\_from}_{\texttt{WG}}$, $\texttt{made\_from}_{\texttt{WP}}$ and $\texttt{is\_a}_{\texttt{GrapP}}$ interchangeably with the lengthier $\texttt{made\_from}\colon \texttt{Wine} \longrightarrow \texttt{Grapes}$, $\texttt{made\_from}\colon \texttt{Wine} \longrightarrow \texttt{Plants}$ and $\texttt{is\_a}\colon \texttt{Grapes} \longrightarrow \texttt{Plants}$, respectively, and similarly for other duplicate link names. A theory has *axioms*, forming a set of statements held as the basic truths of the theory, from which others can be proven as theorems of the theory. Commutative diagrams express the axioms and theorems of the theory. By specifying certain diagrams to be commutative in our categorical completion, we decree axioms; other commutative diagrams result from this, yielding the theorems. Decreeing that all finite coproducts and pullbacks exist provides an additional axiom scheme, yielding more theorems, and specifying certain of these to be formed from objects and arrows which were nodes and links in the graph yields additional axioms. In this way, the ontology is made to have a rather rich semantics. To pay for this, however, a software system for ontologies that accepts the graph and specified categorical structures and axiom schemes as input must enforce the overall specification of these items in its operation. A system of this kind constitutes a *categorical inference engine* that provides for not only in-

ferencing over the ontology, but as we shall see, the corresponding operations on data repositories which can be incrementally updated via operations executed by the users through the facets.

To summarize, Figure 4 shows one of the commutative diagrams that exists simply because all compositions exist in a category. In effect, it is a *definitional* axiom stating that $\texttt{made\_from}_{\texttt{WP}}$ has the same meaning as $\texttt{is\_a}_{\texttt{GrapP}} \circ \texttt{made\_from}_{\texttt{WG}}$. Given the understanding that $\texttt{is\_a}_{\texttt{GrapP}}$ is intended to indicate a subentity relationship, categorically a *subobject*, and that $\texttt{made\_from}_{\texttt{WG}}$ has its intended meaning as "made from", then $\texttt{made\_from}_{\texttt{WP}}$ also has the meaning "made from" because the only change is that the "G" part of $\texttt{made\_from}_{\texttt{WG}}$ is a subentity of the "P" part of $\texttt{made\_from}_{\texttt{WP}}$. But so far the "subentity" and "made from" notions are intuition, not formalization. Formalizing the intuition is the subject of subsections 2.2–2.6.

2.2 Terminal objects, global elements, and coproducts

Categorical completions of a graph can be extended in many ways depending upon what is to be assumed about the resulting theory. One extension we have made is to declare that certain diagrams in addition to those given by composition commute as in Figure 4. An additional extension is the inclusion of a terminal object in **Bev**; its usefulness will become apparent presently. Now, category theorists often overload suggestive symbols such as 1 in many ways for economy of notation; the context is expected to clarify the intended meaning. In the present case, $\mathbf{1}$, written in boldface, denotes the terminal object. A terminal object in a category has the property that, for every object x, it serves as the codomain of a unique morphism $!_x \colon x \longrightarrow \mathbf{1}$ having x as its domain. This is illustrated in Figure 5, which shows a graph for a very small category (except that identity morphisms $\mathrm{id}_x \colon x \longrightarrow x$ and compositions of the labelled morphisms such as $g \circ f \colon a \longrightarrow c$ are not shown). Again for economy of notation, the subscript on the ! symbol is normally omitted, as shown. Notice that the morphism $! \colon \mathbf{1} \longrightarrow \mathbf{1}$ is not shown, since it is just the identity for $\mathbf{1}$, $\mathrm{id}_{\mathbf{1}} \colon \mathbf{1} \longrightarrow \mathbf{1}$ (remember, the ! morphism for $\mathbf{1}$ is unique, so it must be $\mathrm{id}_{\mathbf{1}}$).

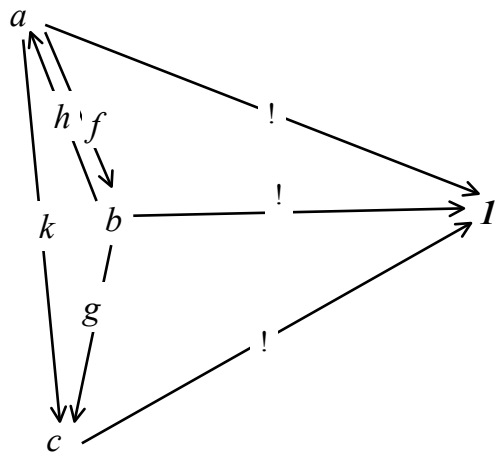


Figure 5: **In a category with a terminal object $\mathbf{1}$, every object x is the domain of a unique morphism $! \colon x \longrightarrow \mathbf{1}$.**

Suppose that in **Bev**, instead of the entities (now regarded as objects) `Grapes`, `Grains`, and `Plants`, we have

only Plants. A morphism whose domain is **1** is called a *global element*. Given that we have a terminal object, we could replace the entities Grapes and Grains by global elements grapes: $\mathbf{1} \to$ Plants and grains: $\mathbf{1} \to$ Plants. Global elements play a role in categories similar to (yet significantly different from) that of set-theoretic elements. In the present case, which is the classifying category of a theory, global elements are the constants expressed in the theory which enable reasoning over specified instances. Making them global elements of an entity instead of entities in their own right changes the logical status of grapes and grains in the theory, but we can retain the Grapes and Grains entities along with the global elements. In particular, this will ensure that the diagram of Figure 4 remains as part of the theory.

Now **Bev** has two arrows with domain **1** and codomain Plants. As suggested in the discussion of missing links in Section 2.1, let us include in the ontology the arrow made_from: AlcoholicDrinks $\longrightarrow$ Plants. Now we can regard Plants as an attribute of Alcoholic Drinks (the made_from attribute) with solely the two values grapes and grains by specifying that it is one of the finite coproducts in **Bev**. The base diagram for this coproduct is the discrete diagram D_1 in Figure 6 consisting of two copies of **1**. The coproduct cocone consists of the two arrows grapes: $\mathbf{1} \to$ Plants and grains: $\mathbf{1} \to$ Plants. Since D_1 is discrete, there is no requirement that triangles formed by the cocone and diagram morphisms commute, since there are none. We shall make use of Plants as an attribute in subsection 2.4.

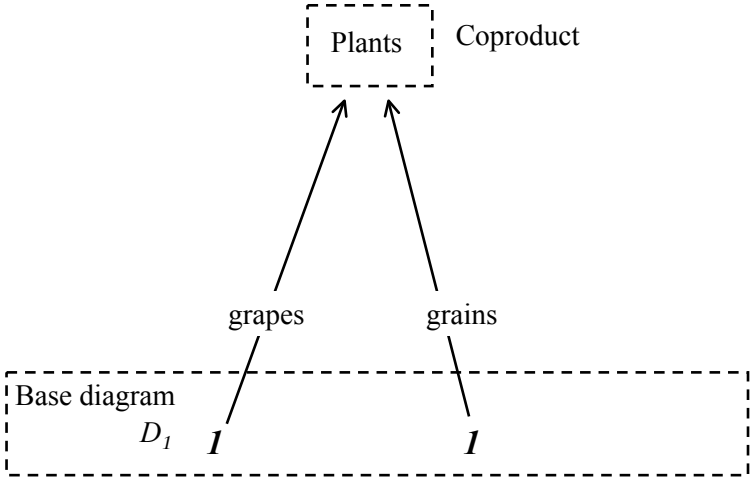
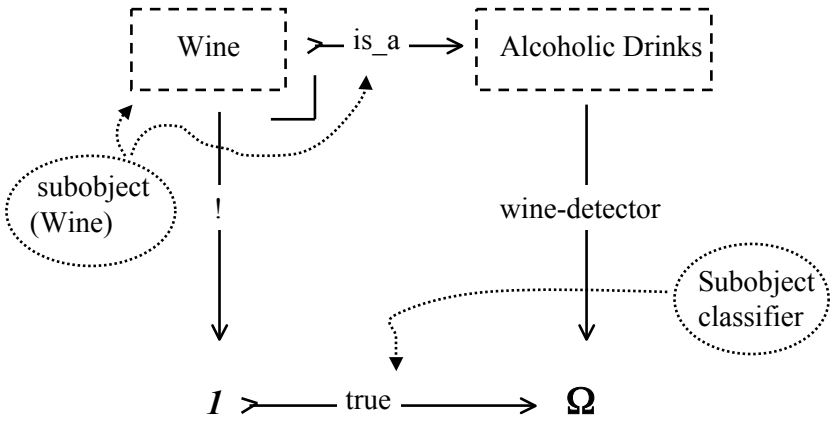


Figure 6: **The global elements** grapes **and** grains **and their common codomain** Plants **form a cocone over a discrete diagram consisting of two copies of 1.**

At this point, it is worth showing with a rather detailed argument that global elements are *monics*. This will establish an important fact firmly in mind and also illustrate reasoning in category theory. A monic in any category is a morphism $m\colon a \to b$ with the following property: If $p\colon c \to a$ and $q\colon c \to a$ are any two morphisms such that the compositions $m \circ p : c \to b$ and $m \circ q\colon c \to b$ are equal, in other words if $m \circ p = m \circ q$, then it is the case that $p = q$. But for all objects x, **1** is the codomain of a *unique* morphism having x as its domain. This means that if $m\colon \mathbf{1} \to b$ is an arbitrary global element, *any* two morphims $p\colon c \to \mathbf{1}$ and $q\colon c \to \mathbf{1}$ must both be $!\colon c \to \mathbf{1}$ and therefore we have both (1) $m \circ p = m \circ ! = m \circ q$ and (2) $p = q$. This shows that global elements are monics.

Monics are normally signified by attaching tails to the arrows. In Figure 6 the tails were omitted. They will



" Wine is an alcoholic drink"

Figure 7: **A pullback defining the entity** `Wine` **as a subentity or in categorical terms a subobject of** `Alcoholic Drinks`**.**

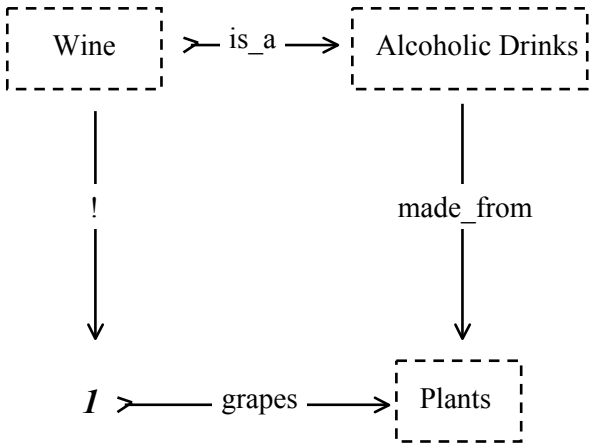
be present in subsequent figures, while in the text monics will be indicated only by stating that they are monics.

2.3 Subentities as subobjects

As mentioned earlier, the `is_a` links in an ER graph suggest a subclass relationship between entities. This means that if we have a notion of "instances" of entities, then corresponding to every instance of `Wine` there is a unique instance of `Alcoholic Drinks`, so that `Wine` instances form a subclass of the class of `Alcoholic Drinks` instances. *Subobjects* are a type of categorical construct that appear in many useful categories and yield a notion of subclass in a classifying category. In the category **Set** of sets and functions, subobjects are subsets, a kind of subclass. In general, a subobject in a category is a monic (actually, a class of monics, but calling it a single monic will suffice for our purpose). In many very important categories, there is an object known as a *subobject classifier* that generalizes the notion of selecting a value from the set of boolean values $\{0, 1\}$ (again overloading numerical symbols). By definition, a subobject classifier $\texttt{true}\colon \mathbf{1} \longrightarrow \Omega$ (provided one exists) has the property that for all monics $m\colon a \longrightarrow b$ in the category, there exists a morphism $\Phi\colon b \longrightarrow \Omega$ such that $m\colon a \longrightarrow b$ and $!\colon a \longrightarrow \mathbf{1}$ form a pullback cone for the diagram consisting of Φ and `true`. Figure 7 shows an example of the use of a subobject classifier in representation, where m is the $\texttt{is\_a}_{\texttt{W\_AD}}$ morphism and Φ is the `wine-detector` morphism. In the category **Set**, Ω is the set $\{0, 1\}$ and the terminal object $\mathbf{1}$ is the singleton $\{0\}$. The global element `true` is a function which maps the element 0 of $\mathbf{1}$ to itself as an element of Ω. The function $m\colon a \longrightarrow b$ expresses the subset relation $a \subseteq b$, and Φ is the characteristic function Φ for the subset a of b. The same is true in the category **FinSet** of `finite` sets and functions, with which we will actually be working: $\Omega = \{0, 1\}$, the terminal object of interest $\mathbf{1} = \{0\}$, and so forth. In the interpretation of the diagram of Figure 7, the `wine_detector` function is the characteristic function for the `Wine` subset of the `Alcoholic Drinks` set—alcoholic drinks that are wines are mapped to 0 by this function and all others to 1. In this way, a subobject classifier formalizes the notion of an `is_a` morphism. *An* `is_a` *morphism is the pullback of the subobject classifier* $\texttt{true}\colon \mathbf{1} \longrightarrow \Omega$ *along a morphism* $\Phi\colon b \longrightarrow \Omega$. In particular, an `is_a` morphism is a monic.

2.4 Applying Categorical Semantics

With is_a links properly formalized as described in the preceding section, they can be used in queries. One example of this is a type of inferencing that constructs entities "on-the-fly" that were not defined in the ER graph. To be specific, pullbacks of global elements in our ontology can be used to formalize a version of the database Select operation that finds a subobject associated with a particular attribute value. Before we show how this works, let us formally re-formulate the made_from links from Beer to Grains and from Wine and Champagne to Grapes as compositions involving the newly-added made_from link from Alcoholic Drinks to Plants. That is, we require that the diagram in Figure 8 and similar diagrams for champagne and beer commute. This diagram imposes the constraint that the composition of is_a: Wine $\rightarrow$ AlcoholicDrinks with the made_from link is essentially the global element grapes. The unique morphism !: Wine $\longrightarrow$ **1** can be thought of as "translating" the domain Wine of the composition grapes$\circ$! to the domain **1** of grapes. From this and Figures 6 and 7, we can then infer that all instances of the subclass Wine are made from grapes, and also make the appropriate "made from" inferences for champagne and beer.

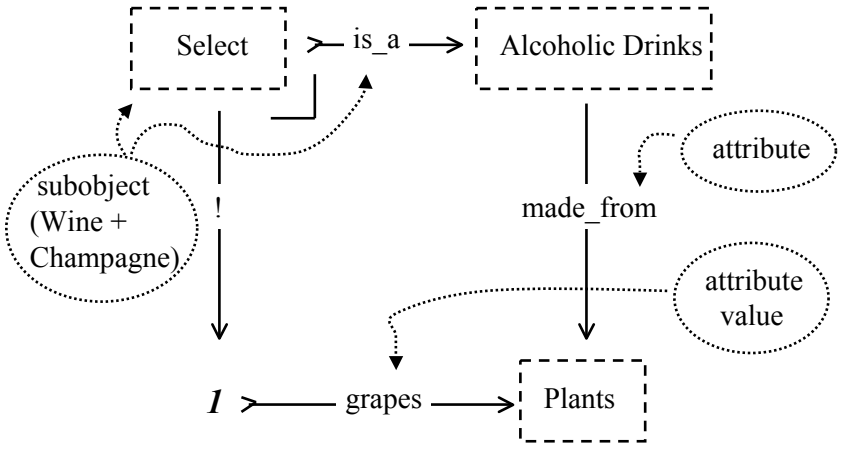


"Wine is made from grapes"

Figure 8: **A diagram specifying that wine is made from grapes. In a model of the ontology, this diagram serves as a constraint upon data entry (for example, during a data base insert update).**

The software based upon the formalization can then search through the appropriate subobjects to form the pullback cone of Figure 9. This illustrates the database Select operation under the specification that the instances of Alcoholic Drinks have a specified attribute value. Consider the made_from attribute for Alcoholic Drinks, which we have agreed is now a single made_from link whose codomain is the attribute value entity Plants. As shown in Figure 9, applying Select to find the alcoholic drinks that are made from grapes can be specified by a pullback involving the global element (attribute value) grapes. The pullback object Select is the domain of an is_a morphism, which we can claim really is an is_a because, as in the case of the is_a morphism in Figure 7, it is the pullback of a monic, hence is itself a monic. Recall that at the beginning of this section we specified that pullbacks exist in **Bev** for all diagrams with pairs of morphisms having a common codomain, as do grapes: **1** $\longrightarrow$ Plants and made_from: AlcoholicDrinks $\longrightarrow$ Plants. In any model of the ontology in the category **FinSet**, therefore, the Select set contains exactly those Alcoholic Drinks elements that are made from grapes. Since the Wine and Champagne entities appear in commutative diagrams such as that in Figure

8, and are, we suppose, the only ones that do so together with the global element `grapes`, the `Select` subset contains all `Wine` and `Champagne` elements and nothing else.



"Find the alcoholic drinks made from grapes"

Figure 9: **The `Select` operation as a pullback.**

2.5 Repository states as models

We have made reference to interpreting the objects and morphisms of the ontology as finite sets and functions. These are the objects and morphisms of the category **FinSet**, a subcategory of the category **Set** of all sets and functions. We use the term "model" in this context. This is because we formalize repository states as collections of data organized in concert with the ontology, and we do this by regarding the organization as a system of finite sets and functions, the traditional way of regarding the models of a theory in formal logic. The entities of our ontology are the sorts of the theory (or predicates, if we wish to use unsorted theories), expressed as objects of the category **Bev**. The links between entities are operations of the theory, expressed as morphisms of **Bev**; the symbolic instances of the entities are constants of the theory, expressed as global elements in **Bev**. In a traditional set-theoretic model of a theory, each sort is associated with a set, each operation symbol is associated with a function from one set to another, and each constant is associated with an element of its designated set. Unlike the traditional formal logic notion of a model, however, in categorical logic and model theory a model is a functor $D\colon \mathbf{E} \longrightarrow \mathbf{C}$ that preserves certain designated categorical structures. In our case, these are a terminal object and pullbacks (hence, all finite limits), coproducts, and a subobject classifier (recall that all functors preserve commutative diagrams). In our example, **E** is the classifying category of the theory **Bev** and **C** is **FinSet**. Notice in particular that since constants of the theory are global elements in a categorical formulation, which are morphisms, they map to global elements in **C**. Notice that although the category **FinSet** is used for models, as sets and functions are used in traditional model theory, the models in categorical model theory are significantly different from traditional set-theoretic models. For example, in the categorical setting constants are mapped not to set-theoretic elements, but to global elements, which are morphisms.

A central feature of our approach to faceted ontologies is the ability to mathematically characterize updates to a repository that can be made safely, and to design faceted ontologies accordingly. Data can then be entered and

used to provide a useful service to a system of experts communicating through the ontology. The mathematical vehicle for the characterization of safe updates is the system of natural transformations between the functors that formalize the models. Given a repository state D and information (such as new data) leading to an update to the repository, an *insert* update of D, adding in new information, leads to a new repository state D'. We express this as a natural transformation $m\colon D \longrightarrow D'$, a monic in the category $\mathbf{C}^{\mathbf{E}}$ whose objects are functors $D\colon \mathbf{E} \longrightarrow \mathbf{C}$ and whose arrows are natural transformations on these functors. For updates expressed in the category $\mathbf{FinSet}^{\mathbf{E}}$, for each object a of $\mathbf{E}$ there is a component m_a, a function mapping the set $D(a)$ to the set $D'(a)$. As shown in Figure 10, if $f\colon a \longrightarrow b$ is a morphism in $\mathbf{E}$, the diagram of Figure 10 must commute, that is, $D'(f) \circ m_a = m_b \circ D(f)$. In our example, the models are functors $D\colon \mathbf{Bev} \longrightarrow \mathbf{FinSet}$, objects of the category $\mathbf{FinSet}^{\mathbf{Bev}}$.

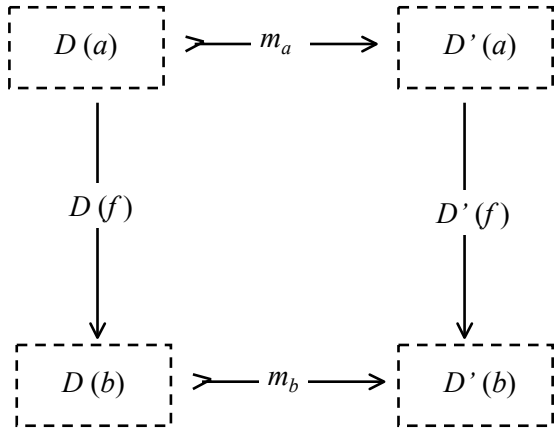


Figure 10: **A commutative square in a repository update expressed as a natural transformation.**

A natural transformation m is a monic if and only if each component m_a is a monic. We can express nearly all updates as inserts and deletions (for example, an in-place update is a deletion followed by an insertion). The components $m_a\colon D(a) \longrightarrow D'(a)$ of a monic natural transformation $m\colon D \longrightarrow D'$ in $\mathbf{FinSet}^{\mathbf{E}}$, being monics in **FinSet**, are injective functions. This expresses the fact that if we insert a new element in $D(a)$, then the set $D'(a)$ has the same elements together with the newly added one. As we shall see in the following sections, updates normally require that some arrows associated with the changed objects must also change. If a and b are objects of $\mathbf{E}$, there are components m_a and m_b and for every arrow $f\colon a \longrightarrow b$ the diagram of Figure 10 must commute, that is, $D'(f) \circ m_a = m_b \circ D(f)$. Inserting an element in the repository as a new instance of concept a while leaving the instance set for b unchanged results in the updated set $D'(a)$ having an element x that did not appear in the previous set $D(a)$. The updated function $D'(f)\colon D'(a) \longrightarrow D'(b)$ must have a new maplet $x \mapsto y$, showing that $y = D'(f)(x)$, for some element y of $D'(b)$ which, since it was not in the update, is the same set as $D(b)$, that is, $D'(b) = D(b)$. This is only possible if the morphism f is included in the update, so that the required maplet can be added to $D(f)$ to make $D'(f)$. A delete update has the form dual to an insert update, that is, it has the form $m\colon D' \longrightarrow D$. For suppose that a single element is deleted from $D(a)$ to obtain $D'(a)$; this can be expressed as a monic component $m_a\colon D'(a) \longrightarrow D(a)$ of m.

Almost all updates to a repository can be expressed as a sequence of delete and insert updates. For example,

assume that in a repository state D, the set $D(\texttt{Wine})$ for the entity `Wine` of **Bev** is a set of strings, as follows:

$$D(\texttt{Wine}) = \{\text{`}\texttt{DomPerignon3000}', \text{`}\texttt{PinotGrigio98}', \text{`}\texttt{SangiovesediGermania}'\},$$

and there is to be an update from this to a final state

$$D'(\texttt{Wine}) = \{\text{`}\texttt{DomPerignon2000}', \text{`}\texttt{PinotGrigio98}', \text{`}\texttt{SangiovesediRomagna}', \text{`}\texttt{Merlot}'\}.$$

Apparently, this update can occur as the result of the following consecutive delete and insert updates:

Delete { `Dom Perignon 3000', `Sangiovese di Germania' }
Insert { `Dom Perignon 2000', `Sangiovese di Romagna', `Merlot' } .

If it were that easy, it could be expressed using the components $m_{\texttt{Wine}}$ and $m'_{\texttt{Wine}}$ of two natural transformations, $m\colon D'' \longrightarrow D$ expressing the delete and $m\colon D'' \longrightarrow D'$ expressing the insert, as follows:

$m_{\texttt{Wine}}\colon D''(\texttt{Wine}) \rightarrowtail D(\texttt{Wine}) =$
$m_{\texttt{Wine}}$: { `Pinot Grigio 98' } $\rightarrowtail$
{ `Dom Perignon 3000', `Pinot Grigio 98', `Sangiovese di Germania' } and
$m'_{\texttt{Wine}}\colon D''(\texttt{Wine}) \rightarrowtail D'(\texttt{Wine}) =$
$m'_{\texttt{Wine}}$: { `Pinot Grigio 98' } $\rightarrowtail$
{ `Dom Perignon 2000', `Sangiovese di Romagna', `Pinot Grigio 98', `Merlot' }

There is just one complication: As has just been pointed out, the D-images of certain arrows involving the entity `Wine` must also be updated. Maplets must be deleted and added to account for elements that have been deleted and added to update $D(\texttt{Wine})$ to $D'(\texttt{Wine})$ via the intermediate set $D''(\texttt{Wine})$. Doing this while satisfying the commutative-square requirements for the natural transformations imposes constraints on updates.

2.6 Relations as spans

As a morphism of **FinSet**, a function $f\colon u \longrightarrow v$ is total: it associates with every element x of its domain u, $x \in u$, a unique element of its codomain, $f(x) \in v$. This is one type of relation, but relations in general do not do this. A relation R on sets u and v is normally expressed as a subset $R \subseteq u \times v$ of the cartesian product of ordered pairs $\{(x, y) \mid x \in u, y \in v\}$. A particularly important case for repositories occurs when some but not all elements of a set have been assigned an attribute value. This can occur, for example, if for some or all models $D\colon$ **Bev** $\longrightarrow$ **FinSet** there occurs in the repository an instance of an alcoholic beverage with no corresponding instance in `Plants` indicating its `made_from` ingredient; this can occur for `Vodka` if something is not done to insert plants other than grapes and grains. Another example is tequila, which is not in the original graph of **Bev**; but what if data for an instance of tequila occurs but $D(\texttt{Plants})$ contains no data for cacti? This might suggest a future revision to the ontology, and therefore it might be advisable to enter any data that becomes available for this or similar beverages. In such cases, for example with $u = D(\texttt{AlcoholicDrinks})$ and $v = D(\texttt{Plants})$, it has often been the practice to interpret R using a *partial function*, for which some elements of u may not be assigned a value of v. For consistency with other morphism interpretations in a model (we have been assuming that they are morphisms of **FinSet**), however , and also for simplicity, it is desirable to use total functions exclusively. One means of addressing this situation and yet employ total functions exclusively is through the use of `NULL`s, a kind of "do-nothing palce-holder element". In this scheme, all $x \in u$ not assigned an attribute value are assigned the value $f(x) = \texttt{NULL} \in v$. But this introduces difficulties, some of them logical; this is discussed in Johnson and Kasangian [8], which cites related investigations of the use of `NULL`s in relational databases. Johnson and Kasangian provide a different strategy in a categorical setting that uses total functions while avoiding the use of `NULL`s. Added advantages of their strategy are that it generalizes to arbitrary relations, and it provides for several types of data repository operations. This strategy we discuss next.

A *span* consists of objects a, b, c in a category and morphisms $h\colon c \longrightarrow a$ and $k\colon c \longrightarrow b$. In Figure 11, the example given in the previous paragraph is reformulated as a span by introducing an entity `SomeADs` and

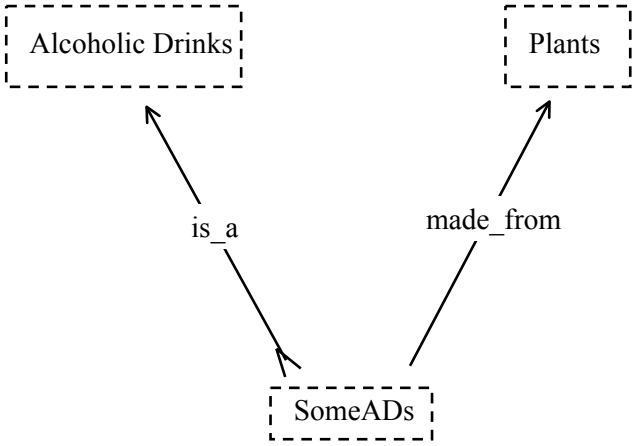


Figure 11: **The partially-defined attribute** `made_from` **of** `Alcoholic Drinks` **expressed as a span involving the subobject** `SomeADs`**.**

a span arrow $\texttt{is\_a}\colon \texttt{SomeADs} \longrightarrow \texttt{AlcoholicDrinks}$, placing a monic in the role of, say, $h\colon c \longrightarrow a$. This is interpreted with a monic (injective) total function $D(\texttt{is\_a})\colon D(\texttt{SomeADs}) \longrightarrow D(\texttt{AlcoholicDrinks})$. The attribute now consists of the other span arrow $k\colon c \longrightarrow b$, which in this case is the reformulated `made_from` arrow $\texttt{made\_from}\colon \texttt{SomeADs} \longrightarrow \texttt{Plants}$, which is interpreted as the total function $D(\texttt{made\_from})\colon D(\texttt{SomeADs}) \longrightarrow D(\texttt{Plants})$. Thus, the attribute can be interpeted as a total function and partiality is expressed through the subobject $\texttt{is\_a}\colon \texttt{SomeADs} \longrightarrow \texttt{Plants}$, where `SomeADs` is the subobject of `AlcoholicDrinks` representing those repository elements of $D(\texttt{AlcoholicDrinks})$ that have been assigned an attribute value. In any update that assigns a value to an element of $D(\texttt{AlcoholicDrinks})$ that does not currently have one, that element can be duplicated in the new (insert-updated) model D' as an element of $D'(\texttt{SomeADs})$. Notice that this also entails updating the two span arrows $D(\texttt{is\_a})\colon D(\texttt{SomeADs}) \longrightarrow D(\texttt{AlcoholicDrinks})$ and $D(\texttt{made\_from})\colon D(\texttt{SomeADs}) \longrightarrow D(\texttt{Plants})$ to map the newly-assigned element of `SomeADs` to itself as the previously-non-assigned element of `Alcoholic Drinks` and assign it the chosen attribute value. Of course, this change to incorporate a span in the ontology entails changes to the original graph to incorporate the span and replace arrows that previously involved `Alcoholic Drinks`, for now they impinge upon `SomeADs` instead. However, in a different ontology for beverages there could have been other arrows involving `Alcoholic Drinks` that would remain unchanged given that they did not involve partially-mapped attributes or other entities related in a non-function way to alcoholic beverages. As shown in [8], other repository operations such as `Join`, `Select` and `Project` with partiality, and composition involving partial attributes, can be expressed through the use of spans and spans together with pullbacks.

2.7 Compatibility with an existing ontology tool

The OWL ontology development system has analogs of some but not all of the categorical constructs presented here. In the syntax of OWL, entities are defined as classes, by a sequence of statements such as the following [16]:

```
<owl:Class rdf:ID="Wine">
    <rdfs:subClassOf rdf:resource="&food;PotableLiquid"/>
```

```
    <rdfs:label xml:lang="en">wine<\/rdfs:label>
    <rdfs:label xml:lang="fr">vin</rdfs:label>
  ...
</owl:Class>
```

Notice the second line of the definition of the class `Wine`; the "`subClassOf`" defines `Wine` as a subclass of `PotableLiquid`, which is in turn a subclass of food. In OWL, every class is a subclass of another, and all are subclasses of `Thing`. In the categorical formalism, this would be equivalent to having every entity be a subobject of a `Thing` entity. This inclusion of all types (entities, classes, "categories") under a single one appears in other ontologies and ontology systems. A further example is John Sowa's concept lattice based on Formal Concept Analysis (FCA) [4, 17]. Although the categorical formalism does not require that there be such an object, in general there is no problem with having a "`Thing` object" with this property. For example, one can easily be included in **Bev**, with the proviso that it cannot be interpreted in **Set** as "the union of all finite sets", which is not finite. On the other hand, we can avoid this difficulty by requiring that it be interpreted as a set containing all things of a particular kind—such as beverages—under the supposition that those things exist in a finite domain. We note in passing the important fact that in expressing a hierarchy of classes in a category with a subobject classifier, if $m\colon a \longrightarrow b$ and $m'\colon b \longrightarrow c$ are monics, then $m' \circ m\colon a \longrightarrow c$ is a monic.

Another feature of OWL is the ability to define "individuals", which are analogous to (but again not equivalent to) our global elements:

```
<owl:Thing rdf:ID="CentralCoastRegion" />

<owl:Thing rdf:about="#CentralCoastRegion">
   <rdf:type rdf:resource="#Region"/>
</owl:Thing>
```

Note that the OWL command `rdf:type` ties an individual to a class of which it is a member. Here, `CentralCoastRegion` is a member of the class `Region`. For our purpose, it is important to note that in OWL this is apparently the only means for representing actual data. This is in contrast to our use of functors in the next section, with which data and relationships between sets of data can be associated as objects and morphisms with the entities and links of an ontology. Also in OWL, there is no notion of a terminal object; hence, there is no mechanism (at least none is visible at present) for using individuals in the same manner as we use global elements in, for example, pullback diagrams as in sections 2 and 4.

The individuals in OWL appear to be analogous to the elements of sets. In the category-theoretic formulation, however, the "individuals" are not elements. A global element is a morphism and, therefore, is interpreted in a model of **Bev** (a functor) by a morphism in the category of sets, that is, by a function. The terminal object of **Bev** is interpeted by a singleton—a terminal object of the sets and functions category—and therefore the interpeted global element *points to* (but is not the same as) a single element of its codomain. For example, the function interpreting `wine`$\colon \mathbf{1} \to$ `AlcoholicDrinks` maps the single element of a singleton to an appropriate element of the set interpreting `Alcoholic Drinks`.

In summary, with respect to "individuals" there are apparently two major differences between OWL and the categorical expression of an ontology. First, the classifying category of a theory has models, functors whose domain is the classifying category and whose codomain is a convenient category such as that of sets and functions. This poses a clear distinction between an ontology and any system of data that grounds its semantics. Second, certain representations of commonly-considered items such as classes and individuals have significant differences in formalization and, hence, require a change in intuition in going from one to the other. For example, a category-theoretic expression of an ontology expresses "individuals" as morphisms of a special kind and, hence, they are not interpreted in the classical sense as elements of a set.

It is possible that there is actually a categorical underpinning for OWL, but it is not evident to us at this writing. What is needed to resolve this is an investigation of the logical foundation of OWL. Also, there exist

alternative systems, such as the Information Flow Framework (IFF) [10] and DAML+OIL [7]. In any case, at this point we do not anticipate any great difficulty in creating an interface between OWL and categorical ontologies as discussed here.

3 Facets and repositories

An ontology is a theory; in our formalization, we express a theory via its classifying category. A theory exists as an object in a category of theories all having the same properties; the arrows of the category are theory morphisms, which are symbol mappings that translate axioms of the domain theory into axioms or theorems of the codomain. In similar fashion, the classifying category $\mathbf{E}$ of a theory exists as an object in a category $\mathcal{T}$ of categories all having certain kinds of structures such as finite limits and coproducts. The category $\mathbf{E}$ expresses the properties of the theory categorically using commutative diagrams together with structures of the kind expressed by all objects of $\mathcal{T}$. The arrows of $\mathcal{T}$ are functors that preserve these structures. In our case, the structures in question are a terminal object and all pullbacks (equivalently, all finite limits), all coproducts, and a subobject classifier as was assumed during the discussion in Section 2. We regard facets as ontologies formalized in the same manner, that is, as categories which are objects in $\mathcal{T}$. For our purpose, a faceted ontology is a system consisting of a number of facets and a main ontology, where each facet is a category together with a functor relating it to the main ontology. For brevity, we often refer to the facet category or functor alone as the facet. Each object of a facet category (a sort of its theory) is mapped to an object of the main ontology category by its functor, and similarly for each morphism (an operation of its theory). Additionally, commutative diagrams are mapped to commutative diagrams as with all functors; however, since it is an arrow of $\mathcal{T}$, a facet-to-main functor must also preserve the structures mentioned: a terminal object, coproducts, pullbacks, and the subobject classifier. Thus, facet functors preserve the same kinds of structure as the model functors described in Section 2.5. In fact, the category **FinSet** is an object in $\mathcal{T}$, and our model functors are morphisms in this category as are the facet functors.

3.1 A formalization of facets

Let us formalize the role of facets. This begins with a definition.

Definition 1. *Let* $\mathbf{E}$ *be a classifying category in* $\mathcal{T}$*. A* facet $\mathbf{V}$ *is a category in* $\mathcal{T}$ *together with a morphism of* $\mathcal{T}$*, that is, a functor* $F\colon \mathbf{V} \longrightarrow \mathbf{E}$ *preserving the aforementioned constructs. We call* $\mathbf{E}$ *the main ontology and, for brevity, may use the terminology "facet" for either* $\mathbf{V}$ *or* F*.*

A facet is the mathematical formalization of a specialized interface for $\mathbf{E}$, the main ontology. We now have a more detailed understanding of the situation illustrated in Figure 3. There, "abx" denotes a sort, an operation, a constant or a formula of the Main Ontology—an object, an arrow, or a commutative diagram of its classifying category $\mathbf{E}$. The notation "abx" indicates that this information in the main ontology is a composite of an "a" and a "b" together with possibly other information, where "a" is the corresponding object, arrow, commutative diagram or other construct common to the objects of $\mathcal{T}$, of Facet 1, which is a category $\mathbf{V_1}$. Similarly, "b" is the corresponding item of Facet 2, or $\mathbf{V_2}$.

The expression of ontologies as categories is meant to express the semantics of the theories in a mathematically concise fashion amenable to computerization. However, a theory is only useful if some of its symbols can be regarded as primitives, whose semantics can be expressed as the kind of things the theory is about. We call this the grounding of the theory. Formally, the theory has models and the grounding is a way of identifying part of a model which, using the inference capability within the theory, can be used to deduce the rest of the model. Regarding a model as a state of a data repository, queries can be addressed in this manner. Also, because the formalization lends itself to symbolic manipulation, for example by calculating pullbacks, symbolic inferencing can also be performed with the ontology. This provides for answering queries in two ways: symbolically, and through data search and other operations performed upon the data repository. Categorically, the inferencing is done with

commutative diagrams and the structures mentioned. Figure 9 is an example of this inferencing using a pullback; it yields a symbolic construct directly in the ontology, and it also yields the associated data from the repository. This favorable circumstance is a consequence of the formalization of a repository state as a model. Categorically, it is a functor into the category **FinSet** that preserves the structures common to all objects of $\mathcal{T}$, where **FinSet** is one of those objects. In terms of computation, it is a data structure with the same entity-relationship organization and categorical constructs as the ontology.

Corresponding to the ER diagram of the Beverage Ontology, let us consider the facet $\mathbf{V_1}$ as the categorical completion of a graph, formed in the same way as **Bev** but with $\mathbf{V_1}$ expressing information on how beverages are made. The full facet includes the functor $F_1 : \mathbf{V_1} \longrightarrow \mathbf{Bev}$. Another facet formed the same way, $F_2 : \mathbf{V_2} \longrightarrow \mathbf{Bev}$, expresses information on how beverages are priced and marketed. Functional relations among these facets are evident by looking at the graph in Figure 1. For instance, in $\mathbf{V_1}$, `made_from: Wine` $\longrightarrow$ `Grapes` expresses the fact that grapes are the principle ingredient in making wine. In $\mathbf{V_2}$, this same arrow indicates that the pricing and marketing of wine is dependent upon that of grapes. To convey this information, the original graphs can be annotated appropriately by attaching descriptions to the entities and information relating the descriptions to the arrows. Annotating with text in this way is an informal device for specializing a facet. A formal means of specializing the facets is to have additonal objects and arrows representing the descriptions in mathematical detail. We discuss this second alternative next; unlike the informal annotation alternative, it is fully consistent with our overall scheme for formalizing faceted ontologies.

To maintain the simplicity of our example, let us consider facet categories formed as subcategories of **Bev**. Notice that in Figure 1, there are links we have not discussed as yet, labelled `costs_less_than`. These become arrows in a categorical completion, such as `costs_less_than: Beer` $\longrightarrow$ `Wine`. In our formal scheme for specializing facets, we can let $\mathbf{V_1}$ contain arrows such as `made_from: Wine` $\longrightarrow$ `Grapes` along with the `is_a` arrows. We can let $\mathbf{V_2}$ contain arrows such as `costs_less_than: Beer` $\longrightarrow$ `Wine` in addition to the `is_a` arrows. Thus, facet $\mathbf{V_1}$ is specialized to information on how beverages are made by showing their ingredients and $\mathbf{V_2}$ is specialized to information on how beverages are priced and marketed by showing their relative pricing in the marketplace. Notice that the two facet subcategories overlap within **Bev**, since they share the `is_a` arrows. Because they express $\mathbf{V_1}$ and $\mathbf{V_2}$ as subcategories of **Bev**, the functors F_1 and F_2 are injective, that is, as morphisms in the category $\mathcal{T}$ they are monics. Thus, facet objects and arrows correspond to objects and arrows of the main ontology as expressed via interface functors. This formalizes the picture in Figure 3.

3.2 Formalizing access to ontologies and data repositories through views

In database applications, data are expected to be shown combined and aggregated in different ways. In relational databases as expressed through, for example, the `SQL` language [3], this is done through operations, or *queries*, including `select`, `join`, `project`, and `union`. In our formalization, the categorical equivalents of these are expressed via the global elements, subobjects, pullbacks, and coproducts that are preserved by the models (functors into **FinSet**). As was mentioned in Section 2, in the closure of an graph giving the basic specification for an ontology, we decree that in addition to any specified pullbacks, coproducts, and so forth, `all` pullbacks, coproducts, etc. exist. This means that in addition to, for example, pullbacks on the appropriate diagrams within the original graph, pullbacks and coproducts involving objects and morphisms of the pullback cones themselves exist in our category, and pullbacks and coproducts on those, etc. Thus, the category **Bev**, although initially expressed in a finite graph along with certain diagrams, cones and cocones consisting of graph nodes and links specifying commutative diagrams, pullbacks and coproducts, is in reality an infinite category. On this account, we cannot maintain **Bev** in its entirety in a computer system (and even if **Bev** were finite, this would be impractical because of the prohibitive work and storage involved). Instead, we compute on demand, on a case-by-case basis, those items (such as the infinite number of pullbacks) that are not in the initial ontology specification. We do this on-demand querying through an additional categorical mechanism: a `view`.

Definition 2. *A view W of a classifying category or ontology $\mathbf{E}$ is a functor $W : \mathbf{W} \longrightarrow \mathbf{E}$, where $\mathbf{W}$ is a classifying category, preserving limits, coproducts, and subobject classifiers. In particular, the interface functors for facets are views.*

Definition 3. *A view W of a view V is a functor $W : \mathbf{W} \longrightarrow \mathbf{V}$, where $V : \mathbf{V} \longrightarrow \mathbf{E}$ is a view of a classifying category* $\mathbf{E}$ *and* $\mathbf{W}$ *is a classifying category, and where W, like V, preserves limits, coproducts, and subobject classifiers. In particular, facets can have views.*

Notice that through composition, a view of a facet can itself be considered alternately as a view or as a (perhaps smaller, more limited) facet. That is, the composition of $W : \mathbf{W} \longrightarrow \mathbf{V}$ with $V : \mathbf{V} \longrightarrow \mathbf{E}$ yields a functor $V \circ W : \mathbf{W} \longrightarrow \mathbf{E}$ having the same property (preserving the structures mentioned in the definitions) as its factors W and V. As a special case, if a view covers an entire facet, only the facet may be necessary to the user as a portal to the ontology. Hence forth, we shall refer to facets and views interchangeably, the emphasis being upon a facet as a view.

Before proceeding further, we remark that the process of enlarging on a categorical closure to include all limits and the other structures mentioned, and also to formalize views and views of views (and views of those ...) is done incrementally beginning with graphs and specifications of particular commutative diagrams, pullbacks, etc. The mathematical justification for this process is given in the very clear presentation of sketch theory in Sections 4.6 and 4.7 of Barr and Wells [2]. Johnson and Rosebrugh [9] recount this in brief, and provide mathematical details relating more closely to the current discussion.

A specialized user who wishes to use a facet for inferencing, for queries addressed to the repository, or for updates of the repository, does so through a view, although as indiciated above we shall regard facets as views and not be concerned with that level of detail. Where necessary, a view can provide a *temporary* or *restricted* interface through which a user is allowed access and through which tasks can be formulated and the results obtained. For example, through a view of the facet $\mathbf{V_2}$ described previously and computerized (in the "categorical inference engine" referred to earlier), a user can perform inferencing, queries and updates. The operations called for in this way can be carried out in the main ontology, which has all the information expressed in $\mathbf{V_1}$ and $\mathbf{V_2}$ combined perhaps with other information. The combined information in the main ontology, together with the data accessible through a portal to the data repository (a computer interface based upon categorical model theory), can provide an accountant with the analytical capability to establish retail prices, or an economist to analyze the market for beverages or commodities used in making them. But how are individual analysts working through their unique facets or views of facets to perform inferencing and data manipulation tasks coherently? For example, if costs must follow new budget constraints through facet $\mathbf{V_2}$, a decision has to be made by the user of facet $\mathbf{V_1}$ to either disallow or include ingredients accordingly through the appropriate updates to repository states. A goal of our investigation will be to find precise conditions for allowing consistent inferencing and data updates through individual analysts' facets.

4 Repository states and updates

The data currently in a repository for a faceted ontology is organized as in a relational database. In our formalization, this is a *state* of the repository. A repository state is expressed as a functor that preserves the previously-mentioned categorical structures. As described in Section 2.5, a repository update is a transition from one state to another, formalized as a natural transformation. The collection of repository states of an ontology $\mathbf{E}$ together with their natural transformations are objects and morphisms in the functor category $\mathbf{FinSet}^{\mathbf{E}}$. Because the states are not only functors but preserve the aforementioned structures, and the natural transformations of interest are monics, they form a subcategory of $\mathbf{FinSet}^{\mathbf{E}}$.

Definition 4. *A* repository state *D of an ontology (classifying category)* $\mathbf{E}$ *is a model of* $\mathbf{E}$ *based in* **FinSet***, in other words, a functor $D : \mathbf{E} \longrightarrow$* **FinSet** *which preserves finite limits, coproducts, and a subobject classifier.*

Definition 5. *An* insert update *(respectively* delete update*) for a repository state D of an ontology* $\mathbf{E}$ *is a monomorphism $D \rightarrowtail D'$ (respectively $D' \rightarrowtail D$) in the category* $\mathbf{FinSet}^{\mathbf{E}}$.

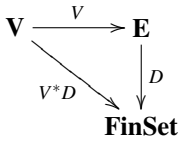
It is stated in [9] that almost all updates to a database through a view can be represented as a combination of insert and delete updates. We shall assume this for faceted ontology repositories, where a facet is formally

identified with a view. These will be the only updates considered in this report, and the exceptions will be addressed in future work.

For any facet (or view) $V : \mathbf{V} \longrightarrow \mathbf{E}$ and any model $D : \mathbf{E} \longrightarrow \mathbf{FinSet}$ it is natural to consider the composite $D \circ V : \mathbf{V} \longrightarrow \mathbf{FinSet}$. Since both functors, hence their composite, preserve the appropriate structures, $D \circ V$ is a model of $\mathbf{V}$. We often omit the composition symbol for functors, and write $DV = D \circ V$. With this notation, the foregoing process yields the *substitution functor*

$$V^* : \mathbf{FinSet}^{\mathbf{E}} \longrightarrow \mathbf{FinSet}^{\mathbf{V}}$$

defined on any repository state D of $\mathbf{E}$ by $V^*D = DV$. We have the following diagram.



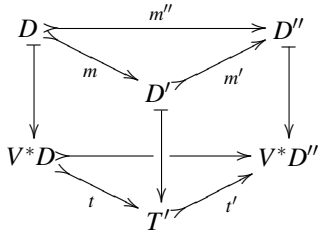
In practical terms, the substitution functor V^* translates a state D of the repository for $\mathbf{E}$ to a state $T = V^*D$ of the repository for $\mathbf{V}$, and likewise for the natural transformations. Now, since states and their updates are automatically translated from the $\mathbf{E}$ to the $\mathbf{V}$ repository, a question arises concerning propagating in the reverse direction: After all, it is to be anticipated that a user will be creating updates to the faceted ontology through that user's facet, which would mean propagating an update from the $\mathbf{V}$ to the $\mathbf{E}$ repository. Further, this propagation must be *safe*, that is, an update made through $\mathbf{V}$ must translate to a valid update to the $\mathbf{E}$ repository, as if it had been made directly through $\mathbf{E}$. Fulfilling these requirements is the point of this section, wherein we discuss *fibrations* (to be defined). A second question of safety arises concerning the direct propagation of updates through V^*. Suppose a repository update by the user of a facet $\mathbf{V_1}$ is propagated safely to the repository $\mathbf{E}$ of the main ontology, where there is a second facet $\mathbf{V_2}$. The $\mathbf{V_1}$ update, having been propagated to the $\mathbf{E}$ repository, will then be automatically propagated to the $\mathbf{V_2}$ repository through V_2^*. Is this second update valid for the $\mathbf{V_2}$ repository?

To address the second question first, note that the propagation of a state D of the $\mathbf{E}$ repository to a state T of the repository for a view or facet $\mathbf{V}$ has the form $T = V^*D = DV = D \circ V$. Therefore, if an object a of $\mathbf{E}$ has the form $a = V(b)$ for some object b of $\mathbf{V}$, the finite set $D(a)$ propagates to a finite set in the $\mathbf{V}$ repository, where the latter set has the form $T(b) = DV(b) = D(a)$. Therefore, a state of the $\mathbf{E}$ repository yields a set in the $\mathbf{V}$ repository if that set, $D(a)$, is the D-image of an object (entity) a of $\mathbf{E}$ which is in turn the V-image of an object (entity) b of $\mathbf{V}$. When this happens, both repositories contain exactly the same set, called $D(a)$ for $\mathbf{E}$-object a and $T(b)$ for $\mathbf{V}$-object b. The same holds for morphisms $f : c \longrightarrow a$ in $\mathbf{E}$: The function $D(f) : D(c) \longrightarrow D(a)$ propagates to the same function, but re-labelled $T(g) : T(d) \longrightarrow T(b)$, in the state T of the $\mathbf{V}$ repository if f, c and a are the V-images of g, d and b, respectively. Thus, a state of $\mathbf{FinSet}^{\mathbf{E}}$ yields identical structures in $\mathbf{FinSet}^{\mathbf{V}}$ for those objects and morphisms that are in the image of V, and therefore states propagate as required. Because the commutative squares that define a natural transformation $m : D \rightarrowtail D'$ in $\mathbf{FinSet}^{\mathbf{E}}$ propagate to the commutative squares defining a natural transformation $t : T \rightarrowtail T'$ in $\mathbf{FinSet}^{\mathbf{V}}$ if the appropriate objects and morphisms are in the image of V, an update to the $\mathbf{E}$ repository propagates safely to the $\mathbf{V}$ repository. Therefore, an insert or delete update to the $\mathbf{V_1}$ repository, if it can be propagated safely to the $\mathbf{E}$ repository, will automatically propagate from there to the $\mathbf{V_2}$ repository through V_2^*. This, then, leaves only the first question unanswered.

Consider an arbitrary facet $\mathbf{V}$. Since $V^* : \mathbf{FinSet}^{\mathbf{E}} \longrightarrow \mathbf{FinSet}^{\mathbf{V}}$ maps repository states and updates in the $\mathbf{E}$-to-$\mathbf{V}$ direction, unless constraints are imposed upon facets there may be no update of the $\mathbf{E}$-repository which would express an update performed through $\mathbf{V}$. Even if there is such an update, there could be many equally-applicable updates, with no clear choice of which to choose. This obviates any chance of automating the facet update process. In database programming, this is known as the view update problem. Fortunately, a universal criterion has been developed for insert and delete view updatability based upon the formalization presented in sections 2 and 3. The key to this solution is to guarantee that the substitution functor V^* is a left and right

fibration. A functor with the fibration property effectively can be "run backward"; that is, a unique morphism can be found in its domain to represent a morphism in its codomain that is in the functorial image. In our case, the morphisms are natural transformations and the fibration property acts as a guarantee that an insert or delete update of the facet propagates to a unique insert or delete update in the main repository of **E**. In what follows we give the definitions of propagatable updates, fibrations, and an even more comprehensive guarantee known as a cofibration.

Definition 6. *Let* **V** *be a facet with* **E** *as the main ontology. Suppose that* D *is a repository state for* **E**. *Let* $T = V^*D$ *and* $t\colon T \rightarrowtail T'$ *be a repository state monomorphism of* $\mathbf{FinSet}^{\mathbf{V}}$ *expressing an insert update of* T. *The insert* t *is* propagable *if* $t = V^*m$ *for an insert update* $m\colon D \rightarrowtail D'$ *in* $\mathbf{FinSet}^{\mathbf{E}}$ *with the following property: for any repository state* D'' *and insert update* $m''\colon D \rightarrowtail D''$ *such that* $V^*m'' = t't$ *for some* $t'\colon T' \rightarrowtail T''$ *with* $T'' = V^*D''$, *there is a unique insert* $m'\colon D' \rightarrowtail D''$ *such that* $V^*m' = t'$, *where the following picture illustrates (with maplets based at the objects) the mapping of the indicated commutative triangle diagram in* $\mathbf{FinSet}^{\mathbf{E}}$ *to one in* $\mathbf{FinSet}^{\mathbf{V}}$*:*



If every insert update on T *is propagable, we say that the facet state* T *is insert updatable.*

The definition of *propagable delete* is dual to this (reverse the arrows and compositions). The definition is a requirement that there be an insert update m mapping to an arbitrary insert t that is essentially unique. That is, if there is another $m''\colon D \rightarrowtail D''$ that maps to an update $t't$ obtainable through a composition involving t, then m'' factors *uniquely* through m via some monic m' that maps to t'. As a special case, the uniqueness guarantees that if $t' = \mathrm{id}_{T'}$ so that $t't = t$, the only alternative is that also $m' = \mathrm{id}_{D'}$ and therefore $m'm = m$. The arrow m in the definition is said to be an *opcartesian arrow* for t and D for the functor V^*. It is a sort of "minimal inverse image" for t because of the unique factorization with an arrow m' that "points toward" any other arrow m'' that has a similar property. This is a way of saying that t can be "mapped backward", because the cartesian arrow m, being universal in the sense of the unique factorization with m', serves unambiguously as a sort of inverse image for it. Another name for opcartesian is *right cartesian*. The definition of *cartesian* or *left cartesian* arrow is dual to the definition of opcartesian or right cartesian arrow and, in the case of the functor V^*, applies to delete updates.

The definition of left and right cartesian arrows applies to functors in general. The following definition follows.

Definition 7. *A functor is a* $\{$*left, right*$\}$ *fibration if every arrow* t *in its codomain has a* $\{$*left, right*$\}$ *cartesian arrow for every object that maps to the* $\{$ *codomain object, domain object* $\}$ *of* t.

Suppose $I\colon \mathbf{A} \longrightarrow \mathbf{E}$ is a fully faithful functor with image closed under isomorphism. The latter condition means that if an object b of **E** is in the functorial image, where $b = I(a)$ for some object a of **A**, then any object of **E** that is isomorphic with b is also in the functorial image. Now suppose further that for any object e of **E** *not* in the image of I it is the case that $\mathbf{E}(e, I(a))$ is empty for every a in **A**. Then I is called a *left cofibration*. A *right cofibration* $J\colon \mathbf{A} \longrightarrow \mathbf{E}$ is a fully faithful functor with image closed under isomorphism such that for any object e of **E** not in the image of J it is the case that $\mathbf{E}(J(b), e)$ is empty for every b in **A**.

We have seen that left and right fibrations guarantee delete and insert updatability, respectively. We end this exposition on faceted ontologies and their associated data repositories with the following lemma. It provides a sufficient condition for a substitution functor V^* as discussed previously to be a left or right fibration. This condition is that the facet interface functor V from which it is derived be a left or right cofibration, respectively. A proof is in [9].

Lemma 1. *If a view or facet* $V\colon \mathbf{V} \longrightarrow \mathbf{E}$ *with main ontology* $\mathbf{E}$ *is a left (respectively right) cofibration, then the substitution functor* $V^*\colon \mathbf{FinSet}^{\mathbf{E}} \longrightarrow \mathbf{FinSet}^{\mathbf{V}}$ *is a left (respectively right) fibration.*

5 Discussion

We have presented a new formalization of faceted ontologies accompanied by data repositories. This formalization is based in category theory, the mathematical theory of structure. Categorical constructs and structural mappings have been shown to have useful properties. They allow the unambiguous expression of the graphical structure of faceted ontologies and their relational data repositories. Mathematical properties of these constructs and mappings, such as diagrams, terminal objects, global elements, pullbacks, coproducts, functors, natural transformations, and fibrations and cofibrations, support inferencing within ontologies and the transfer of knowledge between facets and a main ontology as well as a universal solution to the view update problem, which supports the capability to update repositories through facets.

The applicability of this formalization needs investigation on sample cases of faceted ontologies and their associated data. Further, the formalization needs extension to enlarge upon its capabilites and support complex interactions between facet-specific users and the main ontology and data repository. There are many issues that can arise among the details in implementing the formalization. However, we do not think it useful to attempt to list any such issues at this purely theoretical stage of investigation. This will have to await experimentation with a trial implementation. Some areas of extension can be listed; for example, an initial implementation based upon the formalization must be created given the informal or semi-formal items from an application, such as concept graphs, sets of data, and other items. Again, it is not deemed useful to try to foresee the needed extensions in advance of experimentation with the current formalization. The next phase of this effort will address these needs.

6 Appendix

6.1 Categories

Category theory (see any of [14], [1], [5], [11], and [12]), is based upon the notion of an arrow or morphism, a mathematical relationship between two objects in a category. A morphism reflects the type of mathematical structure of which the objects are exemplars. That is, each morphism $f\colon a \longrightarrow b$ has a *domain* object a and a *codomain* object b, and expresses one of the possible ways in which a relates to b in the context of their common structure. This will become clearer in the examples we shall discuss.

At first glance, the morphisms appear to give a category an underlying directed-graph structure in which objects serve in the role of nodes and morphisms act as edges. There is a defining property that distinguishes categories from graphs, however: the property of *composition of arrows*. In a category $\mathbf{C}$, each pair of arrows having the form $f : a \longrightarrow b$ and $g : b \longrightarrow c$ (where the codomain b of f is also the domain of g as indicated) has a composition arrow $g \circ f : a \longrightarrow c$ whose domain a is the domain of f and whose codomain c is the codomain of g. Composition satisfies two axioms of category theory. The associative law states that in triples which have a head-to-tail match by pairs, $f\colon a \longrightarrow b$, $g\colon b \longrightarrow c$ and $h\colon c \longrightarrow d$, the result of composition is order-independent, $h \circ (g \circ f) = (h \circ g) \circ f$. The identity axiom states that for each object a there is an *identity morphism* $\mathtt{id}_a\colon a \longrightarrow a$ such that for any arrows $f\colon a \longrightarrow b$ and $g\colon c \longrightarrow a$ $\mathtt{id}_a \circ g = g$ and $f \circ \mathtt{id}_a = f$. The notion of composition derives its importance from the fact that many pathways through the morphisms leading from one object to another in a category often yield the same morphism when the composition of the morphisms is calculated along each path. This means that, unlike the situation with graphs, a path through the arrows in a category is associated with a precise notion of cumulative effect or meaning; and, further, different paths whose compositions have the same domain and codomain can have the *same* meaning. When this is true, we have a *commutative diagram* as shown Figure 12. The category in which this diagram is formed is $\mathbf{N}_{|}$; its objects are positive natural numbers, 1, 2, 3, ... , and a morphism from a to b, denoted $|_{a,b}$, exists exactly when

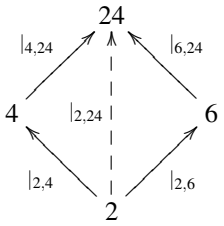
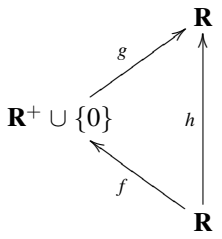


Figure 12: **One of many commutative diagrams in the category $\mathbf{N}_|$.**

a is a divisor of b. There are two apparent morphisms with domain 2 and codomain 24 in the diagram, both being compositions along a path directed through a third diagram object (4 and 6, respectively). Yet, there being at most one divisibility morphism from one natural number to another, we have the equation $|_{4,24} \circ |_{2,4} = |_{2,24} = |_{6,24} \circ |_{2,6}$, stating that the two are equal. More generally, a commutative diagram in any category has the property that any two morphisms having the same diagram objects as domain and codomain, where at least one of them is obtained as the composition of two or more diagram morphisms and the other is obtained in the same fashion or is itself a diagram morphism, are equal.

The category $\mathbf{N}_|$ provides an example in which the morphisms are instances of what we usually think of as a relation—the divisibility relation defined on positive natural numbers. It also exemplifies a relatively simple type of mathematical structure known as a partial order, having at most one morphism between a pair of objects, all morphisms having a shared sense of direction (that is, there are no cycles). In **Set**, which is probably the most familiar example of a category, the morphisms are functions. There are many morphisms in both directions for most pairs of objects in this category, so **Set** is definitely not a partial order. Composition in **Set** is just the familiar composition of functions, with $(g \circ f)(x) = g(f(x))$ for functions $f\colon a \longrightarrow b$ and $g\colon b \longrightarrow c$, with $x \in a$ and $(g \circ f)(x) \in c$. Function composition is associative, and for any set X there is an identity function $\mathtt{id}_X$ whose values are $\mathtt{id}_X(x) = x (x \in X)$, so **Set** is indeed a category. Commutative diagrams in **Set** can be used to express the equivalence of functions, where one of them is obtained as the composition of two or more others. For example, let $f\colon \mathbf{R} \longrightarrow \mathbf{R}^+ \cup \{0\}$, $g\colon \mathbf{R}^+ \cup \{0\} \longrightarrow \mathbf{R}$ and $h\colon \mathbf{R} \longrightarrow \mathbf{R}$ be defined by $f(x) = x^2$, $g(x) = \sqrt{x}$ and $h(x) = |x|$, where $\mathbf{R}$ denotes the set of real numbers and $\mathbf{R}^+ \cup \{0\}$ denotes the nonnegative reals. Then $(g \circ f)(x) = \sqrt{x^2} = |x|$, and the following diagram commutes in **Set**:



One sometimes hears a statement such as "the two [concepts, data types, program constructs, etc.] are in some sense isomorphic", where the term vaguely means "alike in some way". Category theory provides a mathematically rigorous notion of "isomorphism": If a, b are objects of a category $\mathbf{C}$ such that there exist arrows $f\colon a \longrightarrow b$ and $g\colon b \longrightarrow a$ with $f \circ g = \mathtt{id}_b$ and $g \circ f = \mathtt{id}_a$, then the morphism f is called an *isomorphism* (as is g also) and g is called its *inverse* (and f is called the inverse of g), and the two objects are said to be isomorphic. The property of an identity morphism ensures that isomorphic objects in a category are interchangeable

in the sense that they have the same relationships with all objects of the category. An isomorphism in **Set** is a one-to-one, onto function.

An *initial object* of a category $\mathbf{C}$ is an object i having a unique morphism $f\colon i \longrightarrow a$ corresponding to every object a of $\mathbf{C}$. A terminal object t is the dual notion, obtained by reversing arrows in the definition of i—that is, it serves as the codomain of a unique morphism $f\colon a \longrightarrow t$ corresponding to every object a of $\mathbf{C}$. It is easy to show that all initial objects in a category are isomorphic, and the same for all terminal objects. For suppose that i, i' are initial in $\mathbf{C}$; then there must be unique morphisms $f\colon i \longrightarrow i'$ and $f'\colon i' \longrightarrow i$. The compositions $f' \circ f\colon i \longrightarrow i$ and $f \circ f'\colon i' \longrightarrow i'$ must be unique as well, implying that $f' \circ f = \mathtt{id}_i$ and $f \circ f' = \mathtt{id}_{i'}$, which shows that i and i' are isomorphic. The empty set, $\emptyset$, is the single initial object of **Set**, since for any set a there is a unique function $f\colon \emptyset \longrightarrow a$ whose domain is $\emptyset$ and whose codomain is a, namely, the vacuous function, since there are no elements in $\emptyset$ to map to an element of a. There is an infinite number of terminal objects in **Set**, namely the singletons $\{x\}$, since there is a single function $f\colon a \longrightarrow \{x\}$ mapping the elements of any set a to x. One of the most important uses of commutative diagrams and terminal and initial objects is in the definition of a *limit* of a diagram and the dual type of quantity, a *colimit*, two categorical constructs that we use extensively.

6.2 Limits and colimits

Let Δ be a diagram in a category $\mathbf{C}$ as shown in Figures 13 and 14, with objects a_1, a_2, a_3, a_4, a_5 and morphisms $f_1\colon a_1 \longrightarrow a_3,\ f_2\colon a_1 \longrightarrow a_4,\ f_3\colon a_2 \longrightarrow a_4,\ f_4\colon a_2 \longrightarrow a_5$. The diagram $\overline{\Delta}$ in Figure 14 extends Δ to a commutative diagram with an additional object b and morphisms $g_i\colon a_i \longrightarrow b\ (i = 1, \ldots, 5)$, provided additional objects and morphisms with the requisite properties exist in $\mathbf{C}$. That is, $g_1 \circ f_1 = g_2 = g_3 \circ f_2$ and $g_3 \circ f_3 = g_4 = g_5 \circ f_4$. The added cone-like structure K consisting of the *apical object* b and *leg morphisms* g_1, g_2, g_3, g_4, g_5 is called a *cocone* for diagram Δ. In general, a diagram can have many cocones or it can have few or none, depending upon the available objects and morphisms in $\mathbf{C}$. Given cocones K' and K'' for Δ in Figure 13, with respective apical objects b', b'' and leg morphisms g'_i and $g''_i\ (i = 1, \ldots, 5)$, a *cocone morphism with domain K' and codomain K''* is a $\mathbf{C}$-morphism $h\colon b' \longrightarrow b''$ having the property

$$g''_i = h \circ g'_i \quad (i = 1, \ldots, 5). \tag{1}$$

That is, h is a factor under composition of each leg morphism g''_i of K'' with respect to the corresponding leg morphism g'_i of K'. This is illustrated in Figure 13. Re-using the symbol h for notational efficiency, we also denote the cocone morphism determined by h as $h\colon K' \longrightarrow K''$.

With morphisms so defined, and composition of cocone morphisms following directly from composition of $\mathbf{C}$-morphisms, the cocones for Δ form a category, $\mathbf{coc}_\Delta$. A *colimit for the diagram* Δ is an initial object K in the category $\mathbf{coc}_\Delta$. That is, for every other cocone K' for Δ, there exists a unique cocone morphism $h\colon K \longrightarrow K'$. The original diagram Δ is called the *base diagram* for the colimit and the diagram $\overline{\Delta}$ formed by adjoining K to Δ is called its *defining diagram*. Note that, as all initial objects are isomorphic, all colimits for a given base diagram are isomorphic.

The notion of limits in a category can be obtained by dualizing the notion of colimits, that is, by "reversing the arrows" and replacing initial objects with terminal objects. Let Δ be a diagram in a category $\mathbf{C}$ as shown in Figure 15, with objects a_1, a_2, a_3 and morphisms $f_1\colon a_1 \longrightarrow a_3$, and $f_2\colon a_2 \longrightarrow a_3$,. The diagram $\underline{\Delta}$ extends Δ to a commutative diagram with an additional object b and morphisms $g_i\colon b \longrightarrow a_i\ (i = 1, \ldots, 3)$, provided additional objects and morphisms with the requisite properties exist in $\mathbf{C}$. That is, $f_1 \circ g_1 = g_3 = f_2 \circ g_2$. The conical structure K is called a *cone*; note that its morphisms are directed into the diagram, the opposite sense of the leg morphisms of a cocone, which are directed out of the diagram. Cone morphisms are defined appropriately by analogy with cocone morphisms, and again composition follows directly from composition of $\mathbf{C}$-morphisms and the cones for Δ form a category, $\mathbf{cone}_\Delta$. A *limit for the diagram* Δ is a terminal object K in the category $\mathbf{cone}_\Delta$. That is, for every other cone K' for Δ, there exists a unique cone morphism $h\colon K' \longrightarrow K$. Again, the original diagram Δ is called the *base diagram* for the limit and the diagram $\underline{\Delta}$ formed by adjoining K to Δ is called its *defining diagram*. Note that, as all terminal objects are isomorphic, all limits for a given base diagram are isomorphic.

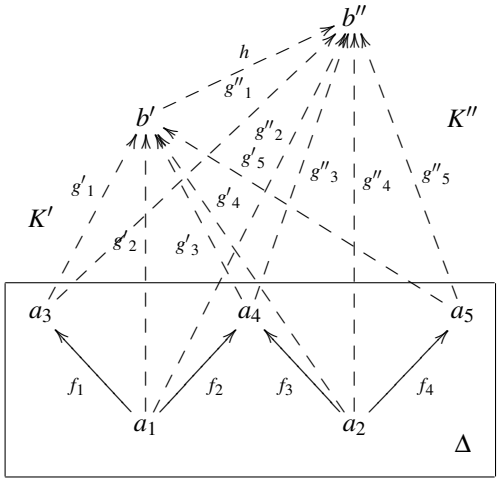


Figure 13: **A cocone morphism $h\colon K' \longrightarrow K''$ in $\mathbf{coc}_\Delta$ is a morphism $h\colon b' \longrightarrow b''$ in C between the apical objects b' and b'' of cocones K' and K'', respectively, that is a factor of each leg morphism $g''_i\colon a_i \longrightarrow b''$ of K'', with $g''_i = h \circ g'_i$.**

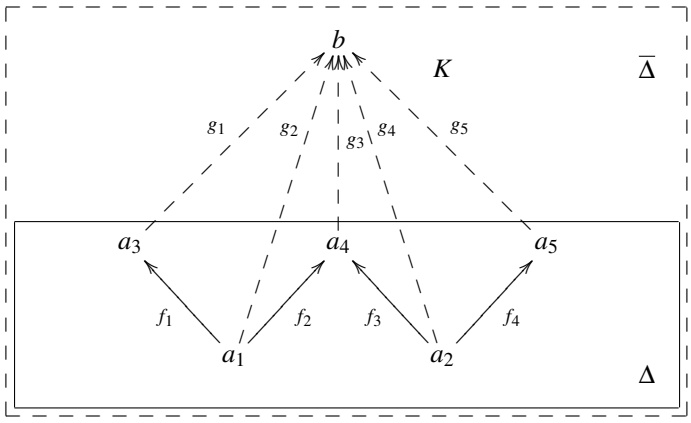


Figure 14: **A colimit for a diagram Δ. The extended diagram $\overline{\Delta}$ extends Δ with a conical structure of morphisms from all diagram objects $a_1, \ldots, a_5$ pointing to an apical object b**

The limit in Figure 15 is an example of a *pullback*. A pullback is a limit for a diagram consisting of three objects and two morphisms (here, f_1 and f_2) sharing one of the objects as codomain. This configuration is called a *cospan* (a *span* is the dual notion—reverse the arrows). Pullbacks are limits commonly used in applications. Johnson and Rosebrugh [9] give examples, and some are given in Section 2 of this report. In categorical logic and model theory, pullbacks often appear along with a terminal object , which is a limit for an empty diagram. Terminal objects can be used to define global elements; one use for these is to express constants in a theory, and they are also useful in defining subobjects [13]; both uses are important in this report. The limits occurring most commonly in categorical logic and model theory are products, which are limits over discrete diagrams. The cartesian product of sets is an example from the category **Set**. Colimits have a history of use in categorical logic and computer science ([6], [19]). A theorem in category theory can be used to derive an algorithm for calculating limits in any category that contains limits for all of its diagrams, and similarly for colimits by dualization (see The Limit Theorem in [14]). Colimits and limits do not exist for all diagrams in all categories, but they can be very useful where they do exist.

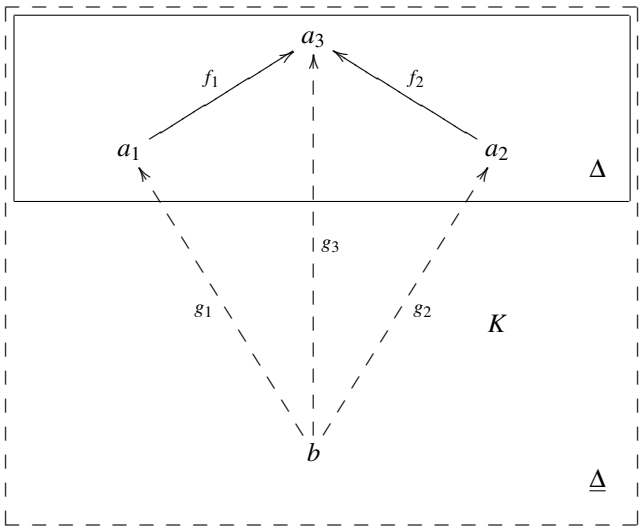


Figure 15: **A limit for a diagram Δ. The extended diagram $\underline{\Delta}$ extends Δ with a conical structure of morphisms to all diagram objects $a_1, \ldots, a_3$ from an apical object b.**

6.3 Structural Mappings: Functors and Natural Transformations

To formalize the association of an ontology with a computational system, we make use of the categorical notion of structure-preserving mappings. These are the essence opf category theory, which studies relationships between different kinds of mathematical systems with a similar underlying structure. The first kind of mapping to discuss is a *functor*, which transports the structure of one category into another category that is capable of expressing it. A functor $F : \mathbf{C} \longrightarrow \mathbf{D}$, with domain category $\mathbf{C}$ and codomain category $\mathbf{D}$, associates to each object a of $\mathbf{C}$ a unique image object $F(a)$ of $\mathbf{D}$ and to each morphism $f : a \longrightarrow b$ of $\mathbf{C}$ a unique morphism $F(f) : F(a) \longrightarrow F(b)$ of $\mathbf{D}$. Moreover, F preserves the compositional structure of $\mathbf{C}$, as follows. Let $\circ_{\mathbf{C}}$ and $\circ_{\mathbf{D}}$ denote the separate composition operations in categories $\mathbf{C}$ and $\mathbf{D}$, respectively (this is for emphasis; normally, the unadorned symbol $\circ$ is used for all categories in a discussion since there is usually no ambiguity). For each composition $g \circ_{\mathbf{C}} f$ defined for morphisms of $\mathbf{C}$, $F(g \circ_{\mathbf{C}} f) = F(g) \circ_{\mathbf{D}} F(f)$, and for each identity morphism of $\mathbf{C}$, $F(\mathtt{id}_a) = \mathtt{id}_{F(a)}$. It follows that F preserves the commutativity of diagrams, that is, the images of the objects and morphisms in a commutative diagram of $\mathbf{C}$ form a commutative diagram in $\mathbf{D}$. This means that any structural constraints expressed in $\mathbf{C}$ are translated into $\mathbf{D}$ and, hence, F is a structure-preserving mapping.

The set of all morphisms with domain a and codomain b (called a *hom-set*) in a category $\mathbf{C}$ is denoted either as $\hom_{\mathbf{C}}(a, b)$ or, more simply, $\mathbf{C}(a, b)$. The intent so far in this discussion has been to avoid foundational issues in mathematics such as Russell's Paradox and the "set of all sets", which lead in the early 20th century to the distinction between sets and proper classes. Notice that there is no guarantee in the general case that the objects or the morphisms in a category form sets as opposed to proper classes; this point is particularly relevant for the category **Set**, to name but one of many common examples. However, the hom-sets $\mathbf{Set}(a, b)$ in **Set** are sets, and this is true of most important categories. These categories are called *locally small*; those whose class of morphisms (hence, also objects) form a set are called *small*. A functor $F : \mathbf{C} \longrightarrow \mathbf{D}$ is *faithful* if it is injective (that is, one-to-one) on each hom-set $\mathbf{C}(a, b)$. It is *full* if it is surjective (that is, onto) each hom-set $\mathbf{D}(c, d)$ that is in the functorial image, that is, where $\mathbf{D}(c, d) = F(\mathbf{C}(a, b))$ for a pair (a, b) of objects in $\mathbf{C}$. A functor that is both full and faithful (in which case the mapping from $\mathbf{C}(a, b)$ to $\mathbf{D}(c, d)$ is bijective, or one-to-one and onto) is often referred to as a *fully faithful* functor.

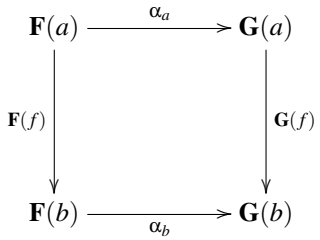


Figure 16: **A commutative diagram associated with a natural transformation. The morphisms $\mathbf{G}(f)\circ\alpha_a : \mathbf{F}(a) \longrightarrow \mathbf{G}(b)$ and $\alpha_b \circ \mathbf{F}(f) : \mathbf{F}(a) \longrightarrow \mathbf{G}(b)$ are one and the same, $\mathbf{G}(f)\circ\alpha_a = \alpha_b \circ \mathbf{F}(f)$.**

Not only are there structure-preserving mappings between categories, but also structure-preserving relations between the mappings themselves. A *natural transformation* $\alpha : F \longrightarrow G$ with domain functor $F : \mathbf{C} \longrightarrow \mathbf{D}$ and codomain functor $G : \mathbf{C} \longrightarrow \mathbf{D}$ consists of a system of $\mathbf{D}$-morphisms α_a, one for each object a of $\mathbf{C}$, such that the diagram in $\mathbf{D}$ shown in Figure 16 commutes for each morphism $f : a \longrightarrow b$ of $\mathbf{C}$. That is, the morphisms $G(f)\circ\alpha_a : F(a) \longrightarrow G(b)$ and $\alpha_b \circ F(f) : F(a) \longrightarrow G(b)$ are actually one and the same, $G(f)\circ\alpha_a = \alpha_b \circ F(f)$. In a sense, the two functors have their morphism images $F(f) : F(a) \longrightarrow F(b)$, $G(f) : G(a) \longrightarrow G(b)$ "stitched together" by other morphisms α_a, α_b existing in $\mathbf{D}$, indexed by the objects of $\mathbf{C}$. In plain Engtlish, the composition of the morphisms along the two paths leading from one corner $F(a)$ of a commutative square to the opposite corner $G(b)$ yields the same morphism, independently of the path traversed.

References

[1] Jiri Adamek, Horst Herrlich, and George Strecker. *Abstract and Concrete Categories*. Cambridge University Press, 1990.

[2] Michael Barr and Charles Wells. *Category Theory for Computing Science*. Prentice-Hall, second edition, 1995.

[3] Donald D. Chamberlin and Raymond F. Boyce. Sequel: A structured english query language. In *Proceedings of the 1974 ACM SIGFIDET Workshop on Data Description, Access and Control*, pages 249–264. Association for Computing Machinery, 1974.

[4] Richard Cole, Peter Eklund, and Gerd Stumme. Document retrieval for email search and discovery using formal concept analysis. *Applied Artificial Intelligence*, 17(3), 2003.

[5] Roy L. Crole. *Categories for Types*. Cambridge University Press, 1993.

[6] J. A. Goguen and R. M. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the Association for Computing Machinery*, 39(1):95–146, 1992.

[7] Ian Horrocks. DAML+OIL: a description logic for the Semantic Web. *Data Engineering*, 25(1):4–9, 2002.

[8] Michael Johnson and Stefano Kasangian. A relational model of incomplete data without `nulls`. In *Conferences in Research and Practice in Information Technology(CRPIT)*, volume 109, pages 89–94. Australian Computer Society, 2010.

[9] Michael Johnson and Robert Rosebrugh. Fibrations and universal view updatability. *Theoretical Computer Science*, 388(1–3):109–129, 2007.

[10] Robert E. Kent. *The Information Flow Framework (IFF)*. IEEE, 2003. Located at http://suo.ieee.org/IFF/slides.pdf .

[11] F. W. Lawvere and S. H. Schanuel. *Conceptual Mathematics: A First Introduction to Categories*. Cambridge University Press, 1995.

[12] Saunders Mac Lane. *Categories for the Working Mathematician*. Springer-Verlag, 1971.

[13] Saunders Mac Lane and Ieke Moerdijk. *Sheaves in Geometry and Logic: A First Introduction to Topos Theory*. Springer-Verlag, 1992.

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

[15] Roberto Poli and Leo Obrst. The Interplay Between Ontology as Categorical Analysis and Ontology as Technology. In Roberto Poli, Michael Healy, and Achilles Kameas, editors, *Theory and Applications of Ontology: Computer Applications*, chapter 1, pages 1–26. Springer, 2010.

[16] Michael K. Smith, Chris Welty, and Deborah L. McGuinness, editors. *OWL Web Ontology Language Guide*. The World Wide Web Consortium (W3C), 2004. Located at http://www.w3.org/TR/owl-guide/ .

[17] John F. Sowa. *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks Cole Publishing Co, Pacific Grove, CA, 1999. See also the KR Ontology discussion at http://www.jfsowa.com/ontology/.

[18] Christopher Welty. Ontology research (editorial). *AI Magazine*, (Fall 2003):11–12, 2003.

[19] Keith E. Williamson, Michael J. Healy, and Richard A. Barker. Industrial applications of software synthesis via category theory–case studies using specware. *Automated Software Engineering*, 8(1):7–30, 2001.