Sketches, Views and Pattern-Based Reasoning

Ralph L. Wojtowicz
Baker Mountain Research Corporation (ralphw@bakermountain.org)
Shepherd University (rwojtowi@shepherd.edu)

***Abstract*—The mathematical theory of sketches provides a graphical framework for describing and relating knowledge representations and their models. Maps between sketches can extract domain-specific context from a sketch, express knowledge dynamics and be used to manage representations created for distinct applications or by different analysts. There are precise connections between classes of sketches and fragments of first-order, infinitary predicate logic. EA sketches are a particular class that is related to entity-attribute-relation diagrams and can be implemented using features available in many relational database systems. In this paper we illustrate sketch theory through development of a simple human terrain model. We apply the theory to an example of aligning sketch-based knowledge representations and compare the approach to one using OWL/RDF. We describe the computational infrastructure that is available for working with sketches and outline research challenges.**

I. Introduction

We use the term *knowledge representation* to refer to a mathematical model of the concepts that we use to understand, reason about and navigate our environment. It evolves in response to new experiences, concept formulation and the mission at hand. Ownership, membership, amicability, people and plans are examples of interrelated entities in this network. We use *decision space* to refer to a sets of individuals and relationships that our knowledge representation organizes. This space is more dynamic, densely populated and uncertain than the knowledge representation. The concept of ownership, for example, encompasses a list of ephemeral connections between individuals and their possessions. Our understanding of ownership persists while instances of this relationship come and go. Moreover, different people can share a common understanding of ownership even if the instances of this relationship that they observe have little or no overlap. They apply the same knowledge representation to distinct models.

Different knowledge representations may characterize the same concept in distinct ways. Renaming the concept 'ownership' as Eigentum or propriété, for example, results in a new presentation of the concept. A complex idea may, more generally, be decomposed into distinct, simpler concepts by different people. Finally, as we build a knowledge model to organize our observations of a greater range of phenomena, we frequently derive and extract parts of it that are suitable for context-based reasoning about particular situations.

A mathematical formulation of knowledge should distinguish between the knowledge representation and decision space models. It should support evolution of the former and the dynamics and uncertainty that are characteristics of the latter. The mathematical framework should support derivation of context-specific views of a knowledge representation and a decision space. Finally, it should provide mechanisms for aligning knowledge models that differ due to simple renaming and more complex reformulations of concepts.

Examples of knowledge representations include relational algebra and its implementation in database languages (such as SQL), entity-relation-attribute diagrams, ontologies, data specifications and sketches. Our interest in the latter results from its graphical nature, deep connections between sketch theory and logic and a rich notion of contextual view of a sketch. Sketch theory has proved to be a valuable tool in mathematical logic and the theory of computer programming languages. Its relationship to other semantic technologies, therefore, warrants further exploration.

The purpose of this paper is to introduce the sketch data model to researchers and practitioners of other semantic-based technologies and to describe a program for its application. We seek to give an overview of the theory through discussion and examples without focusing on the mathematical details. The following themes emerge. (1) An ontology or sketch is a *presentation* of knowledge. Different presentations of the same knowledge are possible. The *theory* of a sketch is the formal mathematical object that such presentations generate. (2) Alignment of distinct knowledge representations and derivation of views of particular ones are more appropriately formulated using theories than presentations. (3) The sketch model emphasizes the distinction between a knowledge representation and its models. Instances, incompleteness and uncertainty may be more appropriately incorporated in models rather than in knowledge representations themselves. (4) The software infrastructure available for working with sketches currently is meager compared to that which has been developed around other semantic technologies such as OWL/RDF.

A. Concept of Operations

Figure 1 illustrates an example concept of operations that shows how the sketch data model might be used in a decision support system. Later in this paper we discuss details of particular aspects of the data pipeline. Data from distributed sources is marshaled into local data models $\mathcal{S}$ that are expressed as sketches. The local sketches are aligned using sketch maps into a common parent $\mathcal{T}$ called a *theory*. $\mathcal{T}$ is the sketch generated by the local sketches taking into account potential overlaps. Parent sketches evolve over time as local ones are modified and new data sources come online. Within a particular mission context, a view $\mathcal{V}$ of the system knowledge representation $\mathcal{T}$ is extracted. The problem of mathematically characterizing the context from event and decision histories is a challenging one and is an active area of research for applications such as Internet search. A *view* is then a sketch equipped with a sketch map into the current parent theory. Models of sketches (including views) are distinct from the sketches themselves.

They include the observed instances that populate the classes and relations that symbols in the sketch represent. Uncertainties and partial information are accounted for in the model, not in the sketch. Data artifacts relevant to a view are analyzed to estimate statistical metrics for potential future states. Figure 1 is conceptual and necessarily incomplete. It does not show, for example, the roles of user interface components, visualization tools and query and reasoning engines, nor of event and decision history archives which would be built into a real command decision system.

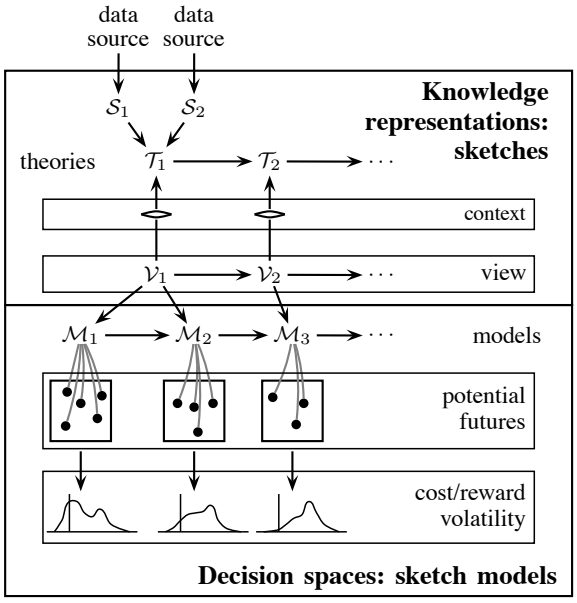


Fig. 1. Concept of operations for the sketch data model

B. Historical Background

Category theory is a mathematical field introduced by Eilenberg and Mac Lane in the 1940s to manage transformations between certain geometric and algebraic structures. It saw explosive growth after Kan discovered the unifying concept of *adjoints* in 1958 [22]. During subsequent decades it has been applied across diverse areas of computer science and mathematics including statistics [9], linguistics [8], dynamic systems, semantics of programming languages [3], topology and, in particular, logic [20] where it provides a non-set-theoretic foundation for mathematics. The theory of sketches is a subdomain of category theory developed by C. Ehresmann in 1968 [11]. It was almost exclusively a tool of the French school of category-theorists until publication of [2], [3]. A category is a collection of *objects* (e.g., sets, probability spaces or vector spaces) and maps between them (e.g., functions, stochastic matrices or linear transformations). In this paper we use categories to construct models of sketches. Sketches themselves form a category having rich structure [13].

II. Sketches, Models and Maps

A sketch is a graph-based knowledge representation. It consists of an underlying directed graph G together with extra structures that impose semantic constraints on models. Figure 2 shows part of the underlying graph of a simple sketch of human terrain knowledge. The vertices Person, Foreign, Coalition, Resident, Village and TribalElement represent classes of entities. Individuals who populate these classes are typically not represented in G (although it is possible to include them explicitly). They instead occur in semantic models of the sketch and may be realized as, for example, rows in database tables. The SeenIn vertex represents a relation between the two classes to which it has edges. It represents the situation in which foreigners may be observed in one or more local villages. Instances of this relation, like individuals who populate the classes, occur in models of the sketch instead of being represented in the sketch itself.

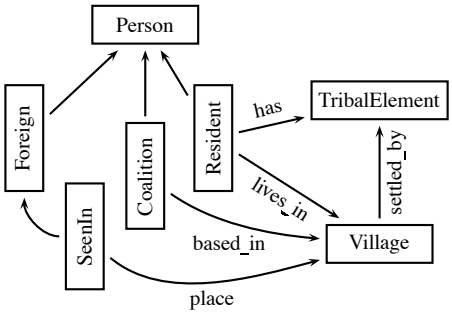


Fig. 2. Part of a graph representing human terrain knowledge

Intuitively, the edges of G represent functions. As we discuss below, however, edges may model incomplete or uncertain information. We intend the edge from Resident to Village, for example, to model a situation in which each resident of an area of interest is associated with a unique home village. Each instance of the class represented by the Coalition vertex is to be based in a specified village. Moreover, each village is settled by a unique tribal element. The two edges from SeenIn represent the process of identifying a foreigner and a village in which he or she was observed. Multiple or no observations of a particular individual are possible. Note that in a sketch, relations (properties) are modeled by vertices rather than edges as is the idiom in OWL/RDF. A relation vertex, however, is the domain of edges that specify the types of entities that it links. That is, the types of the variables that occur as the domain and range of (binary) relations must be specified.

Various features of the human terrain that we seek to represent are not captured by the graph alone. We express these using extra structures called *diagrams*, *cones* and *cocones*. In Section II-A below we give a general discussion of the way in which these constraints are specified using graph maps. In this paper we describe examples of how these concepts are used but do not define them precisely. For details, see [2].

The triangle involving Resident, Village and TribalElement is an example of a *diagram*. It expresses the intuition that the tribal element of a resident R should coincide with the tribal element that has settled the village in which R lives. This semantics is imposed on models of the sketch by including an appropriate diagram in the sketch constraints and by the mathematical definition of sketch model. The constraint can be implemented, for example, using database triggers if the instances are stored in database tables.

The Foreign, Coalition and Resident classes are to be construed as subclasses of Person. Again, this intent is not captured by the graph alone. We express subtype relations by

including particular *cone* constraints in our formulation of the sketch. One such cone would be included for each of the three subtypes that occurs in Figure 2. Cones, like diagrams and cocones, impose mathematical requirements on models.

Finally, we may intend the classes Foreign, Coalition and Resident to be mutually exclusive and to exhaust the possible classifications of Person instances. This feature is not captured by the graph but can be included in the sketch using three cones (to assert the subtype constraints) and a *cocone* to assert the disjoint union constraint.

A. Sketch Maps

A *map* $H \to G$ from a graph H to a graph G is a pair of functions that assigns a G vertex to each H vertex and a G edge to each H edge in a way that respects the source and target information for edges in the two graphs. Graph maps play important roles in defining and applying sketches. First, each of the three types of semantic constraints (diagrams, cones and cocones) is defined as a type of graph map from a base graph B to the underlying graph G of the sketch.

$$B \longrightarrow G$$

The three classes of constraints are distinguished by the shapes of their base graphs. Second, maps between sketches are defined to be maps between the underlying graphs that preserve the semantic constraints. To illustrate this idea, observe that a graph map

$$G \longrightarrow G'$$

between the underlying graphs of two sketches $\mathcal{S}$ and $\mathcal{S}'$ converts each $\mathcal{S}$ constraint $B \longrightarrow G$ into a graph map

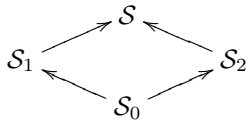
$$B \longrightarrow G \longrightarrow G'$$

via composition of graph maps. If $G \to G'$ is a sketch map, then this composite is also an $\mathcal{S}'$ constraint. Maps between sketches give a rigorous, general framework for addressing knowledge model dynamics, alignment and views. For example, if a knowledge model $\mathcal{S}'$ subsumes another model $\mathcal{S}$, we can express this fact using a sketch map.

$$\mathcal{S} \longrightarrow \mathcal{S}'$$

Not every sketch map expresses a parent-child relationship, however. Those that do are called monomorphisms and satisfy a condition that generalizes the notion of a one-to-one function. A sketch map can, alternatively, merge distinct vertices or edges to eliminate redundancy such as equivalent classes that have been given distinct names.

Alignment of intersecting sketches $\mathcal{S}_1$ and $\mathcal{S}_2$ in a common parent $\mathcal{S}$ is expressed by the following diagram of sketch maps



where $\mathcal{S}_0$ is a sketch representing the intersection of the two knowledge models. We discuss an example in Section II-E.

B. Semantic Models of Sketches

Individuals who populate the classes of a sketch are not typically represented in the sketch itself. They are elements of *models* of the sketch. As we discuss below, this framework clarifies the distinction between the syntax of a knowledge representation and its semantics. We can use this formulation to introduce partiality (i.e., missing data) and uncertainty into models rather than requiring these features to be part of the syntax. First, however, we describe deterministic, set-based models. A (set-based) *model of a graph* G is an assignment of a set $M(v)$ to each vertex v of G and a function $M(e) : M(A) \to M(B)$ to each edge $e : A \to B$ of G. There are no further restrictions on models of a graph.

A *model of a sketch* $\mathcal{S}$ is a model of its underlying graph G that satisfies the restrictions that are represented by the constraints (diagrams, cones and cocones) of $\mathcal{S}$. In this paper we seek to give an overview of the theory and do not give a precise definition of these constraints or their semantics. For details, see [2]. Figure 3 shows a model of a fragment of the human terrain sketch that was shown in Figure 2. The Resident and TribalElement vertices are interpreted as sets of instances. The edge labeled 'has' is interpreted as a function between the sets. To constitute a model of the sketch, the functional interpretations of the edges lives_in and settled_by must be consistent with that of 'has'.

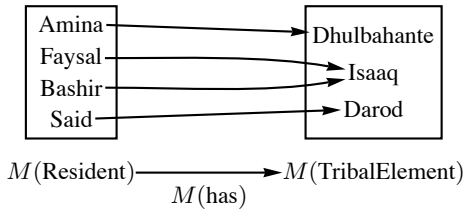


Fig. 3. Functional model of a fragment of the sketch shown in Figure 2.

By varying the semantic category in which sketch models take their values, we may represent lack of information and uncertainty. The edges of the underlying graph may, for example, represent partial functions rather than total functions. Recall that a partial function from a set X to a set Y is a function $X' \to Y$ for some subset of X and that composition $g \circ f$ of partial functions is associative (like composition of total functions) and is defined by further restricting the domain of f. Figure 4 shows a partial function model of a fragment of the human terrain sketch that was shown in Figure 2. In this example, the tribal element membership of Faysal is unknown.

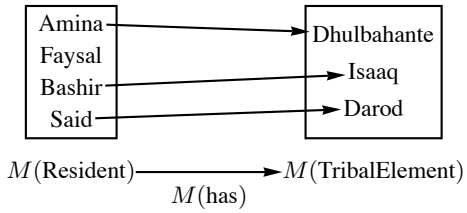


Fig. 4. Partial functional model of a fragment of the Figure 2 sketch.

In Figure 5 we illustrate a probabilistic model of the edge 'has' that occurred in Figure 2. In this model, each point of the source object (which for 'has' is the set M(Resident))

is mapped to a probability function on the target object (the set $M(\text{TribalElement})$ in this case). That is, in this semantic category, edges are interpreted as stochastic matrices (all entries are non-negative and columns sum to 1). Composition is matrix multiplication.

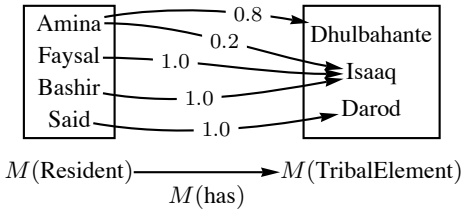


Fig. 5. Probabilistic model of a fragment of the Figure 2 sketch.

There is a rich literature investigating classes of sketches, as distinguished by the types of semantic constraints they include, and classes of semantic models. Examples include linear, finite product, finite limit, EA (entity-attribute) and mixed sketches. The expressiveness of the class of sketch imposes requirements on the classes of structures that may be employed in semantic models. Just as various OWL dialects are associated with different fragments of the predicate calculus, so too are classes of sketches.

C. Maps of Models

The theory of sketches also provides a notion of maps between semantic models. We call these *model maps*. They can be used to represent model dynamics, comparisons and combinations. For example, the fact that different people may populate our tabulations of the Resident class that is represented by the corresponding vertex of Figure 2, should not require us to change the syntax of our knowledge representation. In other words, our understanding of the concepts and relationships of the human terrain does not necessarily change when we observe a new individual to add to our information system. This modular approach to information and knowledge management is a strength of the sketch framework.

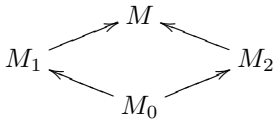
As with models themselves, model maps can introduce partiality and uncertainty. We focus on deterministic maps. Let M and M' be models of a sketch $\mathcal{S}$. For each vertex v of $\mathcal{S}$, the models have corresponding sets $M(v)$ and $M'(v)$ of individuals. A map τ from M to M' is a collection of functions

$$M(v) \xrightarrow{\tau_v} M'(v)$$

between these sets of instances. In order to be a map of models, these functions must be consistent with the functions in the models themselves that arise from edges in the underlying sketch graph. Two models M and M' of the Figure 2 sketch, for example, each have associated sets of Resident and Village instances. If M' subsumes M by, for example, adding new residents, then the new model should maintain the data about the previously-known residents. This is expressed by the following diagram in which we use τ to denote the two functions τ_{Village} and τ_{Resident}.

$$\begin{array}{ccc} M(\text{Resident}) & \xrightarrow{\tau} & M'(\text{Resident}) \\ {\scriptstyle M(\text{lives\_in})}\downarrow & & \downarrow{\scriptstyle M'(\text{lives\_in})} \\ M(\text{Village}) & \xrightarrow[\tau]{} & M'(\text{Village}) \end{array}$$

The definition of map between models requires the two paths to define the same function. That is, the village of a resident who occurs in both models should be the same in both models. Of course, not every map of models represents an extension or subsumption relationship. As with alignment of sketches, we can express alignments of models using maps. For example, alignment of intersecting models M_1 and M_2 in a common parent M is expressed by the following diagram of model maps where M_0 is a model representing the intersection.



D. Presentations and Theories

A sketch (or an OWL ontology) is a compact *presentation* of the much larger body of knowledge $\mathcal{T}$ that it generates. For example, if an ontology defines a class A, a subclass A' of A and a property P that is defined on A, then we can derive a subproperty P' by restricting P to A'. This restriction P' may or may not be explicitly defined in the ontology. It is part of the larger body of knowledge $\mathcal{T}$ that the ontology is designed to present. Sketch theory defines and provides tools for analyzing this generated body of knowledge.

The *theory* $\mathcal{T}$ of a sketch $\mathcal{S}$ is the sketch that $\mathcal{S}$ generates by recursive application of the constructions supported by the type of sketch. These constructions can include property chains (i.e., composition), property inverses (i.e., reciprocals), property restrictions, products (ordered pairs) and coproducts (unions) of classes, and extraction of subclasses and subproperties. The constructions are specified as types of diagrams, cones and cocones since these are the concepts used to specify semantic constraints in sketches. $\mathcal{T}$ is usually much larger than $\mathcal{S}$. It can be infinite even if $\mathcal{S}$ is finite. Consequently, when we write down a knowledge representation, we almost never write down $\mathcal{T}$. We formulate a presentation $\mathcal{S}$ instead.

In Figure 6 we compute a small example. The underlying graph G of the sketch has two vertices and two edges. To make the example a bit more concrete, assume that P represents a class of people and E represents a class of elected officials who serve political districts. The edge r represents an assignment of elected officials to people while u identifies elected officials as particular instances of people. We impose one semantic constraint: the property chain (composite $r \circ u$) of the two edges indicated in the triangle should coincide with the identity function on elected officials. That is, each elected official serves his or her own political district. The finite graph G could, potentially, generate an infinite family of property chains: $r \circ u$, $u \circ r$, $u \circ r \circ u$, $r \circ u \circ r$, etc. The semantic constraint has the effect of truncating this list so that the only distinct properties are those shown in the path graph on the right side of Figure 6. The path graph is the underlying graph of the theory $\mathcal{T}_1$ of the sketch $\mathcal{S}_1$ whose underlying graph is shown on the left side of Figure 6 and whose only constraint is the diagram shown in the center. The derived edge $u \circ r$ connects each person to his or her elected representative.

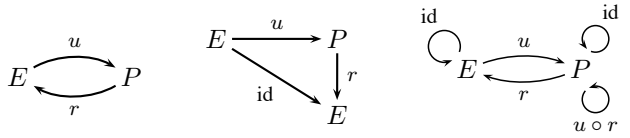


Fig. 6. The two-vertex graph G (left) together with the diagram D (center) generate the theory $\mathcal{T}_1$ (right). The graph and diagram form a sketch $\mathcal{S}_1$.

E. Alignment

A body of knowledge may be presented in different ways even within a fixed formalism (e.g., ontologies or sketches). People use terms differently and use different words to describe the same concepts. The notion of theory of a sketch provides a framework for formulating the alignment problem. Consider the elected officials example discussed above. An alternative presentation is shown in Figure 7. This sketch has only a single vertex C that represents a class of citizens. It has a single edge e that represents the connection of each citizen to his or her elected official. The semantic constraint (indicated by the center diagram) again asserts that each elected official represents himself or herself. The theory generated by this sketch has one vertex and two edges. It is shown below right.

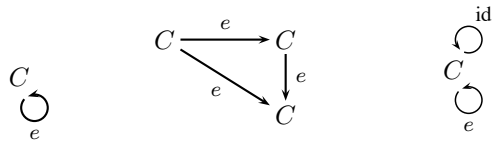


Fig. 7. An alternative formulation of the knowledge model shown in Figure 6. The one-vertex graph G (left) together with the diagram D (center) generates the theory $\mathcal{T}_2$ (right). The graph and diagram form a sketch $\mathcal{S}_2$.

We seek to align these two formulations of the same concepts. To do this we can not use the presentations $\mathcal{S}_1$ and $\mathcal{S}_2$ themselves. We must use theories. Although we can identify the vertex P of the $\mathcal{S}_1$ with the vertex C of $\mathcal{S}_2$, the problem is that there is no edge in $\mathcal{S}_1$ that corresponds to the edge e of $\mathcal{S}_2$. The appropriate edge occurs in the theory $\mathcal{T}_1$ of $\mathcal{S}_1$ not in the sketch $\mathcal{S}_1$ itself. Figure 8 illustrates how the alignment problem is formulated using sketches. The task is to find a sketch $\mathcal{V}$ and sketch maps into the theories generated by the two presentations that we seek to align. $\mathcal{V}$ can be construed as the overlap between the two theories. It is a view (as defined in the next section) of both presentations.

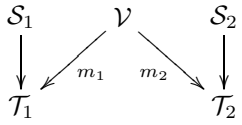


Fig. 8. Formulation of the alignment problem using sketches. To align presentations $\mathcal{S}_1$ and $\mathcal{S}_2$ (i.e., sketches or ontologies), we compute a sketch $\mathcal{V}$ and maps m_1 and m_2 into the theories generated by the presentations.

The sketch framework supports an operation called *pushout* which is essentially the union accounting for overlaps between sketches. With this union operation we can align the two presentations that we have been discussing into a single knowledge representation $\mathcal{T}$. Figure 9 shows the resulting

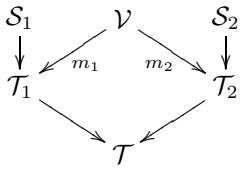


Fig. 9. Alignment of the presentations (e.g., sketches or ontologies) $\mathcal{S}_1$ and $\mathcal{S}_2$ into the common knowledge representation $\mathcal{T}$ using the union (pushout) operation on sketches.

sketches and maps. This illustrates a particular case of the data alignment step shown at the top left of Figure 1.

The need to use structures generated from the available presentations, rather than using the presentations alone, is evident in the alignment example discussed in Chapter 10 of [14]. In this example, two OWL ontologies are aligned using OWL statements. The first ontology is defined below.

```
ex1:Mother       rdfs:subClassOf       ex1:HomeDweller.
ex1:Father       rdfs:subClassOf       ex1:HomeDweller.
ex1:Son          rdfs:subClassOf       ex1:HomeDweller.
ex1:Daughter     rdfs:subClassOf       ex1:HomeDweller.
ex1:hasChild     rdf:type           owl:ObjectProperty.
ex1:hasSon       rdfs:subPropertyOf      ex1:hasChild.
ex1:hasDaughter rdfs:subPropertyOf      ex1:hasChild.
```

The second is defined with prefix `ex2`. It overlaps with but is visibly not equivalent to the first.

```
ex1:Relative     rdf:type                  owl:class.
ex1:Mother       rdfs:subClassOf         ex1:Relative.
ex1:Father       rdfs:subClassOf         ex1:Relative.
ex1:Child        rdfs:subClassOf         ex1:Relative.
ex1:hasParent    rdf:type          owl:ObjectProperty.
```

We align the two using the OWL statements below.

```
ex1:Mother   owl:equivalentClass         ex2:Mother.
ex1:Father   owl:equivalentClass         ex2:Father.
ex1:Son      rdfs:subclassOf              ex2:Child.
ex1:Daugher  rdfs:subclassOf              ex2:Child.
ex1:hasChild owl:inverseOf             ex2:hasParent.
```

Although the `Mother` and `Father` classes coincide, there are no appropriate classes in `ex2` to identify with `ex1:Son` or `ex1:Daughter`. The classes occur in a knowledge representation generated by `ex2` not in `ex2` itself. Similarly, `ex1:hasChild` and `ex2:hasParent` have no corresponding element in the other's ontology. They are identified with elements constructed from the other.

F. Contexts and Views

Decision making uses both general knowledge and specifics of the decision space to balance the expected costs and risks of a program of actions. It focuses on the components that are most relevant to a mission, its goals and tasks. It must efficiently and effectively manage the available data.

Context carries information about intent. Views are implementations of context in a knowledge representation. A *view* of a database is derived using a query. In SQL implementations, a view is typically a single (virtual) table. The *view update problem* addresses the question of how to determine an appropriate update to the state of the total database when a view

is modified. The influential paper [1] developed the *constant complement* approach to view updates. [4], [12] defined the notion of *lens* that characterized a class of updates. At about the same time, [15] described *update strategies* for particular update classes. Sketches were introduced into the study of data semantics in order to better understand database dynamics; in particular, the view update problem [24]. [19] uses sketches to extend the lens concept and classes of view update strategies.

Within the sketch data model, views are particular sketch maps and, in general, are much more expressive than a single derived table. Specifically, a view of a knowledge representation $\mathcal{S}$ is a sketch $\mathcal{V}$ together with a sketch map

$$\mathcal{V} \longrightarrow \mathcal{T}$$

where $\mathcal{T}$ is the theory generated by $\mathcal{S}$ (see Section II-D). Consider, for example, the human terrain sketch shown in Figure 2. One view of interest is obtained by restricting the SeenIn relation to the subrelation in which the village is one in which coalition personnel are based. This view involves subclasses of each of the classes (in addition to the subrelation of SeenIn that we mentioned). None of these subclasses occur in the sketch itself but they are generated by applying cone and cocone constraints.

A challenge to implementing this framework in a decision-making context is using the available event history and other context-specific data to construct an appropriate sketch view in a semi-autonomous manner. The graphical nature of sketches may facilitate the adaptation of recent techniques developed for context sensitive Internet search [6], [16], [21], [28] that use the graph structure of the Web.

G. Support for n-ary Relations

A limitation of OWL/RDF described by practitioners is its lack of direct support for representing n-ary relations. Such properties can be represented directly in sketches. To express a relation R that may hold among n entities that have types $A_1, \cdots, A_n$, we introduce vertices for each of these $n+1$ classes and we include n edges $R \to A_i$. Any axioms that the relation is intended to satisfy would then be formulated using diagrams, cones and cocones.

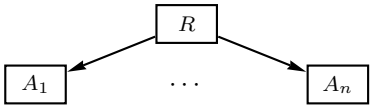


Fig. 10. Sketch for an n-ary relation R among entities of types $A_1, \ldots, A_n$

III. Logical Inference

The graphical nature of the sketch data model supports implementation of pattern-matching reasoning capabilities that emulate the process of experienced decision makers [23]. In classical logic, we express properties and relationships as terms and formulas that are recursively-constructed from basic components. Inference is formulated as rules for deriving valid expressions. Like models of physical phenomena, logics are developed with varying levels of fidelity based on their intended applications. Examples include classical, descriptive, modal and linear logics. Expressiveness, however, comes at the expense of higher computational complexity: inference for first-order logic is undecidable, NP-complete for propositional logic, and P-complete and linear for propositional Horn logic (a property exploited in the Prolog language) [26].

A sketch is an alternative, graphical way of presenting a logical theory [2], [20]. In the sketch data model, we express relationships using diagrams, cones and cocones in directed graphs instead of with formulas and terms. Logical inference employs graph properties associated with constraints. A sketch with no constraints is like a logical signature with no axioms.

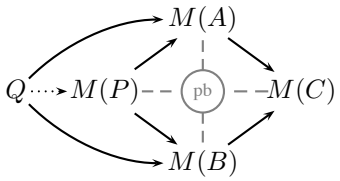


Fig. 11. Universal mapping property that characterizes pullback cones

A pullback cone, for example, is characterized by the property illustrated in Figure 11 which shows a model M of such a constraint (see [2]). If the two outer functions from Q to $M(C)$ are equal, then there is a unique function from Q to $M(P)$ for which the two paths from Q to $M(A)$ are equal as are the two from Q to $M(B)$. Such graph definitions are called *universal mapping properties* [22]. From these we derive other inference rules such as: If the function from $M(B)$ to $M(C)$ is a subtype (`is_a`) relationship, then so is the edge from $M(P)$ to $M(A)$ [22].

Sketches, like logics, are developed with varying levels of fidelity. Linear sketches are the least expressive. Finite limit, finite sum, EA (entity-attribute) and mixed sketches are richer. Despite the distinct character of logical and sketch-based inference, they share deep connections. For various classes of sketches, there are algorithms for constructing logical theories that have equivalent categories of models (see D.2.2 of [20]). Reasoning about a knowledge model expressed as a sketch, therefore, may be achieved either directly using the computational category theory techniques discussed below in IV or indirectly by converting to a first-order theory and using a predicate calculus reasoner.

The problem of pattern-based reasoning with sketches is similar to the ontology alignment problem [18] that is solved mathematically via the *theory* (or *syntactic category*) of a sketch [20]. We may align, for example, the human terrain sketches that are shown in Figures 13 and 2 via a sketch map from the latter to the former. Refinement of the knowledge base to represent levels in a tribal hierarchy (e.g., ethnic groups, tribes, clans and factions) is accomplished with a sketch map from the Figure 2 sketch to a new sketch that would include additional edges and constraints. The simple business knowledge representation shown in Figure 12 can be mapped to a sub-sketch of our human terrain model.

IV. Software Infrastructure

The software infrastructure available for working with sketches is meager relative to that associated for other semantic models (e.g., OWL/RDF). The Easik tool[1] is the most mature.

[1] http://mathcs.mta.ca/research/rosebrugh/Easik

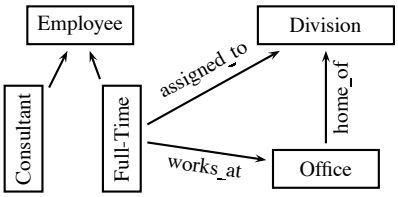


Fig. 12. Part of a sketch representing business structure knowledge

It provides a graphical interface for building a collection of sketches and views. It implements procedures for reading and writing sketches to and from XML and SQL files. It also provides an interface to models maintained in PostgreSQL and MySQL databases. Easik does not implement a reasoning engine. Figure 13 shows a sketch that is similar to the one whose graph is shown in Figure 2. It was developed using Easik. The screen shot illustrates convenient abbreviations for various semantic constraints. The decorated arrows to Person, for example, indicate subtype relationships. These are implemented as cones. Vertices connected to the + symbol form a kind of cocone. Its base consists of the vertices Coalition, Foreign and Resident. The paths connected to CD indicate a diagram constraint.

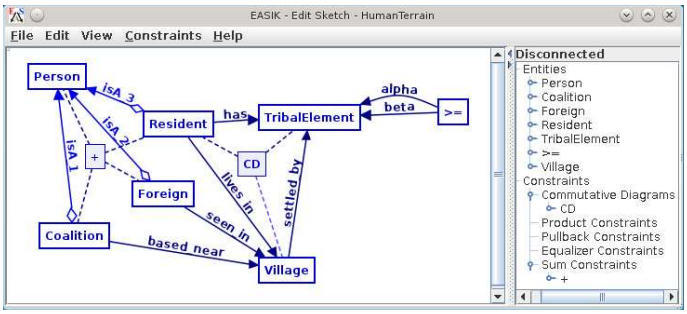


Fig. 13. Human terrain sketch implemented using the Easik software tool

Category theory, despite its abstract nature, is highly computational. All semantic constraints, for example, can be computed from four basic types: cones can be expressed using product cones and equalizer cones; cocones can be expressed using coproducts and coequalizers. One freely-available implementation of these and related computations is written in the ML programming language and is described in [25]. This work and similar research activities could provide a basis for implementing a reasoning engine for a sketch-based information system.

If the only constraints of the sketch are diagrams (i.e., the sketch has neither cones nor cocones), then the theory $\mathcal{T}$ (when $\mathcal{T}$ is, in fact, finite) generated by the sketch may be computed via the left Kan extension algorithm which generalizes the Todd-Coxeter procedure from group theory [7], [27]. If the generated sketch is infinite, the algorithm, of course, does not terminate. Its complexity in the cases when it does terminate has not been characterized and is highly sensitive to small variations in the sketch [5].

V. Implementing Sketch Models in Databases

Features available in major relational database systems including PostgreSQL and MySQL provide an interface between the mathematical theory of sketches and their application. The Easik tool supports read and write operations between knowledge representations and XML files, SQL files and SQL (MySQL or PostgreSQL) database connections. In Easik, a sketch is implemented as a *database schema*. Each sketch entity (graph node) is a *table* created according to the schema. Values that populate the tables form a model of the sketch. Each table has an implicit, integer-valued primary key. In general, a primary key constraint on a table expresses the fact that the values in one or more columns together are a unique identifier of a row. This does not preclude the possibility that two rows may refer, for example, to a single individual. Attributes are table columns. For example, an entity B with no attributes or outgoing edges is implemented in PostgreSQL as follows where the `id` column is an automatically-generated (i.e., `SERIAL`), key.

```
CREATE TABLE B ( id SERIAL PRIMARY KEY );
```

A *foreign key* constraint specifies that the values in one or more columns must match the values occurring in some row of another table. We implement a sketch edge $A \xrightarrow{e} B$ as a foreign key contained in the A-table and referencing the primary key of the B-table. In PostgreSQL this is expressed as follows.

```
CREATE TABLE A ( id SERIAL PRIMARY KEY,
                 e INTEGER NOT NULL REFERENCES
                 B (id) ON DELETE CASCADE
                        ON UPDATE CASCADE );
```

Insertion of a row into the A-table, therefore, involves specifying values for the columns that are introduced into that table for the edges having domain A. Deletion of a B-row can impact the A-table if an A-row references the B-row via the edge $A \xrightarrow{e} B$. Foreign keys serve to implement relations (object properties) in the sketch data model since a relation is simply an entity having edges to the nodes that correspond to the types of its participants.

Although subtype relations (monic edges) are a particular kind of cone constraint, they can be implemented as foreign keys with unique references in the codomain table. In general, however, sketch constraints are implemented using triggers. A *trigger* for a database table or view executes a specified function whenever certain events occur. The simple diagram shown in Figure 14, for example, asserts that semantics of the composite edge f followed by g should equal that of h. In PostgreSQL we express this as follows.

```
CREATE FUNCTION commutativeDiagram0()
      RETURNS trigger AS $commutativeDiagram0$
   DECLARE _cdTarget1 CONSTANT INTEGER := NEW.h;
           _cdTarget2 CONSTANT INTEGER :=
                      (SELECT B.g FROM B
                       WHERE B.id = NEW.f);
   BEGIN   IF   _cdTarget1 IS DISTINCT
                           FROM _cdTarget2
           THEN RAISE EXCEPTION
             'Commutative diagram constraint
              failure';
           END IF;
           RETURN NEW;
   END;
$commutativeDiagram0$ LANGUAGE plpgsql;
CREATE TRIGGER commutativeDiagram0
  BEFORE INSERT ON A
    FOR EACH ROW EXECUTE
      PROCEDURE commutativeDiagram0();
```

It is a trigger that fires before insertion of a row into the A-table to confirm that the values entered in the foreign key columns for f and h satisfy the commutativity constraint taking into account the value in the g-column in an appropriate row of the B-table. Cone and cocone constraints are all similarly implemented. All EA sketch constraints can be constructed from these [22].

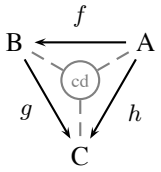


Fig. 14. Diagram to implement using a SQL trigger

A research question that arises is how might one utilize the sketch data model in a context of large-scale, distributed data. Broader demand for a scalable system that supports views and integrity constraints are well-known. Megastore, Tenzing, and Spanner are Google products developed to meet this demand. Apache Cassandra is an open-source alternative.

VI. Conclusion

OWL/RDF and related semantic web technologies have established tenable positions in the intelligence, defense and security domains. The sketch data model, however, integrates a variety of features that can be leveraged. These include its deep connections with infinitary predicate logic, the ability to implement sketches and their models using major relational database systems, its graphical nature and, perhaps most significantly, its sophisticated notion of view of an information system. It is possible to formulate OWL constructs using sketches. The classes of sketches that can be expressed in the dialects of OWL2 is an open question. We have illustrated these concepts and their application to a simple ontology alignment problem.

The sketch data model also clarifies the formulation of certain challenges that we encounter in applications of OWL/RDF. In the sketch approach, uncertainty and lack of information are aspects of models of the sketch. They are not features of the knowledge representation itself. Moreover, sketches (and OWL ontologies) are more appropriately construed as presentations of the larger bodies of knowledge that they generate. In sketch theory, this larger knowledge base is called the theory of a sketch. Except in simple cases of renaming, alignment of presentations involves maps into theories. The extent to which procedures for generating a theory from a sketch can support partial-automation of alignment problems is an open research question.

Finally, the concept of view of a knowledge representation is formulated as a sketch map to a theory. This generalizes the notion of view of a database. Recent techniques developed for context sensitive Internet search exploit the graph structure of the Web and search histories. The extent to which these techniques and the graphical nature of sketches can be exploited to support semi-automated extraction of context-relevant views of a knowledge representation is another open research challenge.

References

[1] F. Bancilhon and N. Spyratos. Update Semantics of Relational Views. *ACM Transactions on Database Systems*. **6**:557–575. 1981

[2] M. Barr and C. Wells. *Toposes, Triples and Theories*. Springer. 1985

[3] M. Barr and C. Wells. *Category Theory for Computing Sciences*. Prentice-Hall. 1990

[4] A. Bohannon, J. Vaughan and B. Pierce. Relational Lenses: A Language for Updatable Views. In *Proc. of the 25th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. ACM. 2006

[5] J. J. Cannon, L. A. Dimino, G. Havas and J. M. Watson. Implementation and Analysis of the Todd-Coxeter Algorithm. *Mathematics of Computation*. **27**(123):463–490. July 1973

[6] H. Cao, D. Jiang, J. Pei, E. Chen and H. Li. Towards Context-Aware Search by Learning a Very Large Variable Length Hidden Markov Model from Search Logs. In *Proceedings of the 18th World Wide Web Conference (WWW'09)*. pp. 191–200. 2009

[7] S. Carmody, M. Leeming, R. F. C. Walters. The Todd-Coxeter Procedure and Left Kan Extensions. *J. Symbolic Computation*. **19**:459–488. 1995

[8] C. Casadio, P. J. Scott and R. A. G. Seely, Eds. *Language & Grammar: Studies in Mathematical Linguistics and Natural Language*. CLSI Publications. 2005

[9] N. N. Čencov. *Statistical Decisions Rules and Optimal Inference*. American Mathematical Society. 1982

[10] E. F. Codd. *Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks*. IBM Research Report RJ599. 1969

[11] C. Ehresmann. Esquisses et types de structures algébriques. *Bul. Inst. Politehn. Iaşi*. **14**:1–14. 1968

[12] J. Foster et al. Combinators for Bi-Directional Tree Transformations: A Linguistic Approach to the View Update Problem. *ACM Transactions on Programming Languages and Systems*. **29**. 2007

[13] J. W. Gray. The category of sketches as a model for algebraic semantics. In *Categories in Computer Science and Logic*. V. 92 of Contemporary Mathematics. AMS. pp. 109–135. 1989

[14] J. Hebeler, M. Fischer, R. Blace and A. Perez-Lopez. *Semantic Web Programming*. Wiley Publishing, Inc. 2009

[15] S. J. Hegner. An Order-Based Theory of Updates for Closed Database Views. *Annals of Math. and Artificial Intelligence*. **40**:63–125. 2004

[16] T. Joachims. Optimizing Search Engines Using Clickthrough Data. *SIGKDD'02, Alberta Canada*. 2002.

[17] M. Johnson and R. Rosebrugh. Sketch Data Models, Relational Schema and Data Specifications. *Electr. Notes Theor. Comp. Sci.* **61**(6):1–13. 2002

[18] M. Johnson and R. Rosebrugh. Ontology Engineering, Universal Algebra, and Category Theory. In *Theory and Applications of Ontology: Computer Aplications*. Springer-Verlag. pp. 565–576. 2010

[19] M. Johnson, R. Rosebrugh and R. J. Wood. Lenses, Fibrations and Universal Translations. *Mathematical Structures in Computer Science*. **22**:25–42. 2012

[20] P. Johnstone. *Sketches of an Elephant*. Oxford University Press. 2005

[21] A. Kustarev, Y. Ustinovesky and P. Serdyukov. Measuring Usefulness of Context for Context-Aware Ranking. *ACM WWW 2012 Companion, Lyon, FR*. 2012

[22] S. Mac Lane. *Categories for the Working Mathematician*. 2nd Ed. Springer-Verlag. 1999

[23] J. Morrison, R. T. Kelly, R. A. Moore and S. G. Hutchins. Implications of Decision Making Research for Decision Support and Displays. In *Decision Making Under Stress: Implications for Training and Simulation*. J. A. Cannon-Bowers and E. Salas, Eds. pp. 375–406. 1998

[24] R. Rosebrugh and R. J. Wood. Relational Databases and Indexed Categories. *Category Theory 1991: Proceedings of an International Summer Category Theory Meeting Held June 23–30*. Vol. 13 of CMS Conference Proceedings. AMS. pp. 391–407. 1991

[25] D. E. Rydeheard and R. M. Burstall. *Computational Category Theory*. Prentice-Hall. 1988

[26] A. Troelstra and H. Schwichtenberg. *Basic Proof Theory*. Cambridge University Press. 2000

[27] R. L. Wojtowicz and N. S. Yanofsky. *Quantum Kan Extensions and Applications*. DOI Contract D11PC20232 Final Report. 2013

[28] B. Xiang, D. Jiang, J. Pei, X. Sun, E. Chen and H. Li. Context-Aware Ranking in Web Search. *ACM SIGIR'10 29–23 July, Geneva, Switzerland*. 2010