Category Theory as a Foundation for Document Processing

Charles Nicholas James Mayfield James Sasaki *

Submitted to Mathematical and Computer Modeling

Abstract

Documents, particularly electronic documents that are created, disseminated, and used with computers, have several representations. Users may wish to work with such electronic documents in any of a document's representations, and this can make it difficult to maintain consistency between the different representations of a document. Category theory provides insight into this problem. We begin by discussing a general abstract semantics for documents. This semantics is made mathematically precise using the language of category theory. We apply the semantics to the problem of maintaining consistency across different representations of an electronic document. The SNITCH hypertext system, which uses semantic nets to represent the meanings of documents, provides a concrete example of this brand of semantics.

1 Introduction

As computers become increasingly important in the preparation, dissemination, and everyday use of documents, it is no longer appropriate to view a document as an object with a single abstract or concrete representation. More often, documents are multi-representational, in the sense that during its existence a single document can take on many different forms. A document that begins, for example, as a loose collection of ideas in a text file, might become a LaTeX input file with embedded images created with a drawing program, a hypertext document broken up into nodes and links,

*Computer Science Department, University of Maryland Baltimore County, Baltimore, MD 21228-5398. {nicholas, mayfield, sasaki}@cs.umbc.edu.

or an output file expressed in Postscript. For multi-media documents, containing video or audio data as well as text and figures, the idea of a single representation for the entire document is even less realistic.

When we consider the many ways in which documents are created and used, it is clear that different representations are suitable for different purposes, but no single representation is adequate for everything. The idea that documents can naturally be manipulated in several representations has been discussed for several years (see, for example, Chen and Harrison [1]). If users were content to work with a single representation of a document, allowing the computer to deal with any other processing, consistency between representations would not be an issue. However, users need to work with all the representations, including (occasionally) the lowest levels of the page description language. In this paper, we argue that the basic ideas of category theory are an appropriate formalism for discussing the multi-representational nature of documents. These ideas provide insight into how document processing systems of the future should be designed, and we offer a system of our own construction as an example. We do not pretend to make any new contribution to category theory *per se*.

In Section 2 we discuss the notion of document, especially the electronic document, in terms of operational semantics. In Section 3 we present a few basic ideas from category theory and describe how they apply to electronic documents. Section 4 addresses the question of consistency between representations. In Section 5, we describe the SNITCH system, an experimental hypertext system based on semantic nets. We pay special attention to SNITCH's multi-representational aspects. In Section 6 we discuss the insights from category theory that are related to the design of SNITCH and to document processing in general.

2 General abstract semantics for documents

What is a document? This simple question is difficult to answer precisely. Some objects are obviously documents: books, papers, magazines; but advances in technology have blurred the distinction between traditional written documents and other forms of presentation, making it at best difficult to define the notion of "document" via physical representation in a given media. Instead, let us try to define the notion of "document semantics".

To help understand the general semantics of documents, consider the semantics of computer programs: programs are certainly documents, and their

semantics have been studied intensively. Three main kinds of semantics are recognized: *denotational*, in which programs are mapped to mathematical objects, *operational*, in which one shows how to abstractly execute a program, and *axiomatic*, in which one discusses relevant properties of the state of computation as one executes a program [2]. These three methods are interrelated: a program might be denotationally mapped to the mathematical object that describes the operational transformation of abstract machine states, where these states are interesting because they satisfy certain predicates.

Combining these notions and applying them to documents, consider the following provisional definition:

> A *document* is an object constructed for the purpose of causing transformations in the state of mind of users to whom the document is communicated.

This proposed definition is too general for our purposes: we are interested in factually-oriented documents, containing text, equations, tables, images, etc. We are not interested in the symbolic, semiotic, and affective content of documents, nor do we wish to consider the treatment of non-written or non-representational forms of communication (abstract sculpture, for example). Thus for our purpose, we make the following definition:

> A *document* is a representational object, containing some combination of text, equations, tables, images and other information, constructed for the purpose of causing transformations in the cognitive/knowledge state of its readers.

3 Using categories as models for documents

To make this notion of document more precise, we will use the language of category theory. The reader who is already acquainted with category theory may skip to Section 3.2.

3.1 Basic Definitions

In this subsection we offer only a few words of explanation of the most basic ideas of category theory. (For more information on this subject, we recommend the works of Barr, Goguen, and Pierce [3, 4, 5].)

A *category* is a collection of objects over which are defined *arrows*, or mappings, that preserve the essential nature or structure of the objects. More formally, a category C comprises [4]:

- a collection of objects;
- a collection of arrows;
- operations that assign to each arrow an object $dom\ f$, its domain, and an object $cod\ f$, its codomain;
- a composition operator that assigns to each pair of arrows f and g, with $dom\ g = cod\ f$, a composite arrow $g \circ f : dom\ f \rightarrow cod\ g$, such that the following *associative law* holds: for any arrows $f : A \rightarrow B$, $g : B \rightarrow C$, and $h : C \rightarrow D$, (with A, B, C and D not necessarily distinct), $h \circ (g \circ f) = (h \circ g) \circ f$);
- for each object A, an identity arrow $id_A : A \rightarrow A$ such that the following *identity law* holds: for any arrow $f : A \rightarrow B, id_B \circ f = f$ and $f \circ id_A = f$.

A number of familiar mathematical objects can be viewed as categories, including sets, graphs, monoids, automata, and abstract data types.

One may also study mappings across categories. Mappings that take objects from a category and produce objects belonging to that same category are "endomorphisms". Mappings may also take objects from one category and produce objects belonging to some other category. If such a mapping preserves the structure of the object (whatever that means in terms of the first category), then that mapping is a *functor.*

3.2 Documents as a Category

For a simple example of a category of documents, define **TS** to have text strings for its objects and editing operations as its arrows; use function composition for arrow composition.

For a more interesting example of a category of documents, let us show that simple hypertext documents consisting of nodes and links (both untyped) form a category **HT**. In this model of hypertext, nodes contain text or other information, and relationships between pieces of information stored in the various nodes are explicitly represented by links connecting those nodes [6]. Because the structure of such hypertexts is so similar to the

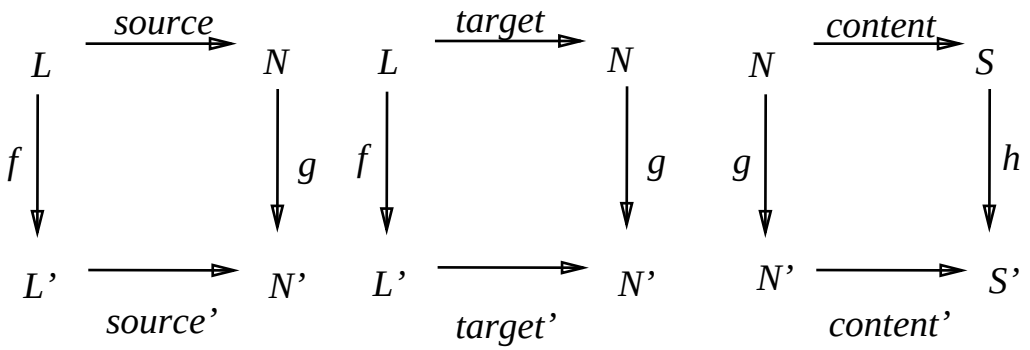

Figure 1: The functions making up a structure-preserving arrow in the category of simple hypertext documents.

structure of directed graphs with labeled vertices, the argument below follows that given by Goguen for directed graphs [3].

Suppose that a simple hypertext H consists of a set N of nodes, a set L of links, a set S of strings representing the contents of nodes, and three functions $source : L \to N$, $target : L \to N$, and $content : N \to S$. Since the major components of these simple hypertexts are sets, the major components of their arrows should be functions that preserve the hypertext structure. A arrow from $H = (N, L, S, source, target, content)$ to $H' = (N', L', S', source', target', content')$ consists of three functions $f : N \to N'$, $g : L \to L'$, and $h : S \to S'$ such that the diagrams shown in Figure 1 commute. (A diagram *commutes* if the indicated mappings can be applied in any order in which they are well-defined, with the same result. For example, for $x \in L$, $source'(f(x)) = g(source(x))$.)

This simple hypertext representation is just one of the many possible representations for a document. There may, for example, be a representation of the document as a string written in a markup language such as SGML, and another representation of the same document as an internal data structure in a particular hypertext system.

If each representation of a document is a member of a certain category, then to give a meaning to a document with respect to a given semantics, we must identify a category of "abstract" documents, and we must give a functor from (at least one of) the various representations of that concrete document to the category of abstract documents. Abstract documents are just another representation, and the nature of these documents depends on

what properties and operations we are interested in. In ordinary text processing, the abstract document is simply an idealized physical representation based on character strings. If the application requires some amount of natural language understanding, the abstract document is an expression of the concepts discussed or used in the document.

The representation used in the abstract document may express the meaning of the concrete document in two ways:

1. The abstract document may simply indicate information to be added to the reader's knowledge. In this case, the notation used to express the abstract document would be a knowledge representation language.

2. Alternatively, the abstract document may be an expression of the comprehension process used by the reader as he or she comes to understand the document. In this case, the meaning of the abstract document is expressed in an operational (as opposed to denotational or axiomatic) fashion.

4 Maintaining consistency between representations

When a user (or a set of users) works with different representations of the same document, maintaining consistency between representations becomes a serious problem. Recall that functors are structure-preserving mappings between categories. If each document representation scheme is a category, then different representations of a document are consistent if and only if the mappings from each representation to the abstract document are appropriately constrained functors.

4.1 Mapping one representation to another

If $\mathbf{C}$ and $\mathbf{D}$ are categories for documents, then let us say that a *representation of* $\mathbf{C}$ *in* $\mathbf{D}$ is a pair of functors $rep : \mathbf{C} \rightarrow \mathbf{D}$ and $unrep : \mathbf{D} \rightarrow \mathbf{C}$ such that $rep \circ unrep = id_{\mathbf{D}}$. (In the language of category theory, *rep* is a split epi, *unrep* is a split monic, and $unrep \circ rep$ is idempotent [7].)

For example, we might represent text string documents in hypertext by taking $\mathbf{TS}$ and $\mathbf{HT}$ for $\mathbf{C}$ and $\mathbf{D}$ respectively. For *rep*, the functor that takes a text string and yields its hypertext representation, we can use a functor *parse*, which presumably involves some natural language understanding. For

unrep, we can use a functor *pprint* that pretty prints hypertext. The *parse* and *pprint* functors are not necessarily inverses of each other, but we insist that whatever text is returned by the pretty printer, it must parse to the same **HT**-meaning as the original text. That is, on objects, $parse = parse \circ pprint \circ parse$. (See Figure 2.) Alternatively, we might take **HT** as our initial representation and use a semantic meaning functor *sem* that maps hypertext to some other, more abstract, category of documents **AD**. In this case, the representation constraint requires $unsem \circ sem = id_{\mathbf{HT}}$. (See Figure 3).

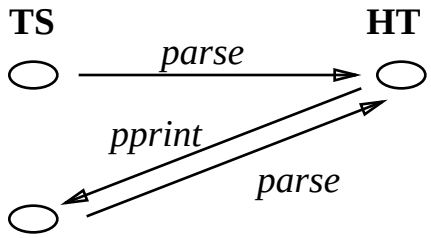

Figure 2: Representation restriction on plain text and hypertext documents.

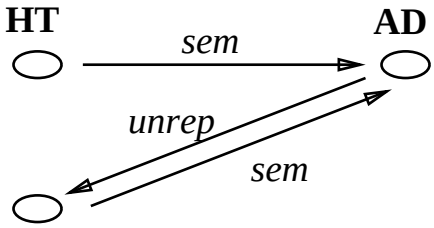

Figure 3: Representation restriction on hypertext and abstract documents.

Composing two representations yields another representation. For example, composing the *parse* and *sem* functors yields a functor that takes text to abstract documents; it must obey the same constraint as above: $sem \circ parse \circ pprint \circ unsem = id_{\mathbf{AD}}$. (See Figure 4.) Note that the requirements in Figures 2, 3, and 4 are all instances of the same general requirement of consistency.

Functors not only map objects to objects, they also map arrows to arrows: if $F : \mathbf{C} \rightarrow \mathbf{D}$ is a functor, then for any arrow $f : A \rightarrow B$ of $\mathbf{C}$, $F(f)$ must be an arrow that maps $F(A)$ to $F(B)$, in $\mathbf{D}$. In terms of abstract and

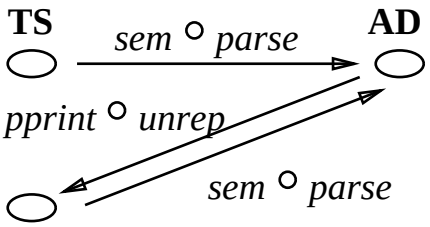

Figure 4: Composition of representation restrictions on plain text, hypertext, and abstract documents.

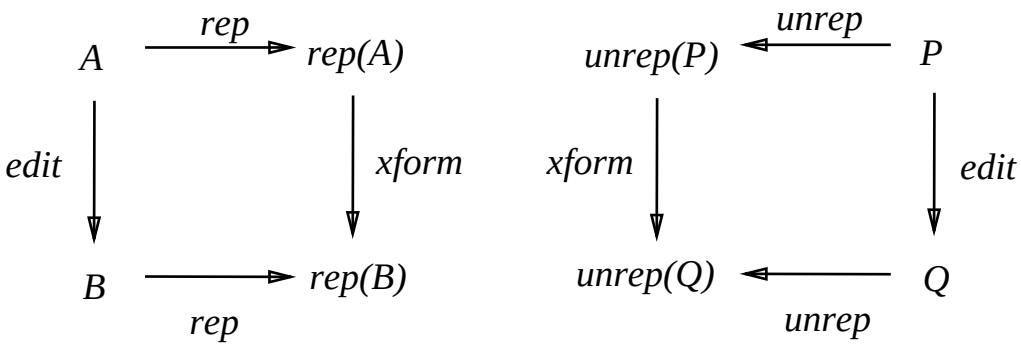

Figure 5: The functoriality of the mappings between document categories guarantees the document's consistency during editing.

concrete representations of documents, the functoriality of *rep* and *unrep* force transformations on documents to be synchronized.

Suppose, for example, that an *edit* mapping is applied to a document representation A in category **C**, resulting in a new document representation B, as shown in the left side of Figure 5. Let *xform* be *rep*(*edit*), then we are guaranteed that *xform* changes the **D**-object representing A to the **D**-object representing B. If *rep*(*edit*) is the identity arrow, then we will say that *edit* is a *formatting change;* it does not affect the meaning (in **D**) of the document. In the other direction, the right side of Figure 5 shows how editing the abstract document corresponds (via *unrep*) to changes in the representation in category **C**.

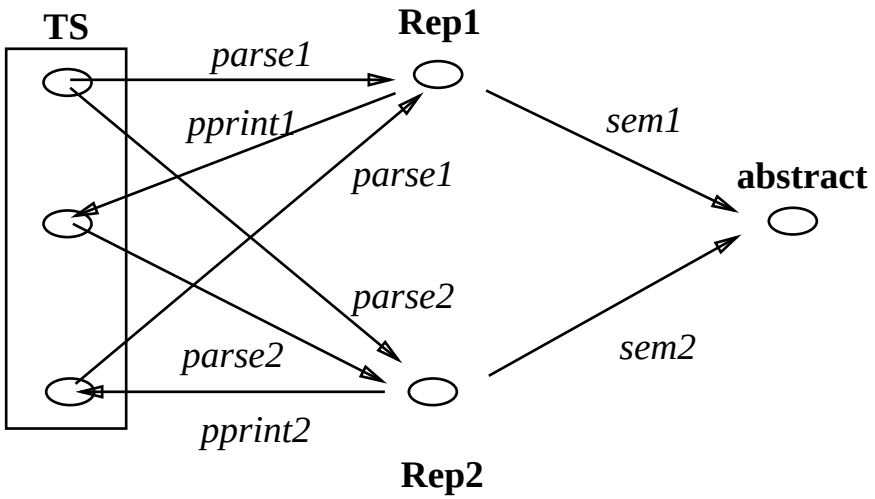

Figure 6: Representation restriction on multiple representations of documents.

4.2 Consistency of multiple representations

Suppose a document processing system maintains multiple representations of the same document. What does it mean for these different representations to be consistent? Say $\mathbf{Rep}_1$ and $\mathbf{Rep}_2$ are two categories used for representing documents, and we have $parse_i$ and $pprint_i$ functors between **TS** and $\mathbf{Rep}_i$ for each i. To discuss consistency of our representations, we also need a category of more-abstract documents **AD**, with functors sem_i and $unsem_i$ between $\mathbf{Rep}_i$ and **AD** for each i. For a particular piece of text to be consistently representable in $\mathbf{Rep}_1$ and $\mathbf{Rep}_2$, we need the diagram in Figure 6 to commute. Roughly, the diagram says that if we parse the text in each of the two different ways, we obtain representations that have the same meaning; in addition, pretty printing the representations yields text that parses to the same meanings. As in the previous section, the functoriality of the $parse_i$ and sem_i means that any editing change on the original text must correspond to some (possibly null) changes to the representations, which each in turn correspond to some changes to the abstract document. We insist that these changes match. That is, for each arrow *edit* in **TS**, $sem_1(parse_1(edit)) = sem_2(parse_2(edit))$.

The restrictions on our representation functors implies that in a document processing system, consistency behind different representations of a

document can be achieved by applying the necessary functors whenever a change is made to a particular representation of the document. Efficiency considerations complicate the situation, however. To rebuild each representation of the entire document whenever any user makes a small change would be inappropriate. A sufficiently large abstract document should be divided into pieces so that locking (of the sort used in traditional database [8]) and update operations can be restricted to relevant portions of the abstract document, and corresponding portions of the various representations. Similarly, the functors that connect the different representations to the abstract document should be implemented incrementally. That is, a small change in one representation should be rapidly reflected in the abstract document, and thereafter in the other representations; these operations should cause minimal disturbance to other users who are working with different parts of the document.

Note that this idea of consistency subsumes the more familiar problem of coordinating read and write access to a database, so that users are prevented from destroying each others' changes to a document. To solve this synchronization problem, access to the abstract document needs to be managed; trivial changes to individual representations of the document that have no impact on the abstract document need not be synchronized.

5 Example of a Multi-Representational Document Processing System

We are exploring the application of natural language processing to multi-representation documents as we develop the SNITCH system [9]. We begin this section with a description of the SNITCH system. We then describe how the ideas from category theory presented above are relevant to the multi-representational character of SNITCH.

5.1 An Overview of SNITCH

SNITCH is a hypertext system in which each document is augmented with a semantic net representation of that document. A *semantic net* is a graph whose vertices correspond to concepts (*e.g.* "person" or "automobile"), and whose edges correspond to relationships between those concepts (*e.g.* "owns"). Semantic nets are widely used to represent knowledge about the world and to express the meaning of phrases and sentences of natural lan-

guage.

The construction of a SNITCH hypertext begins with the definition of a semantic net that describes concepts relevant to the domain of discourse and relationships between those concepts. Because of the domain-specific nature of this knowledge, this semantic net (which will be only one part of the abstract document) is called the *ontological* semantic net. SNITCH uses a semantic net notation that is a variant of the KODIAK knowledge representation language [10].

To incorporate a document into the SNITCH system, a semantic net representation of the meaning of the document, together with connections from words and phrases in the document to the semantic net nodes that represent them, must be constructed. This portion of the semantic net, called the *document* semantic net, can be constructed either manually or automatically. The document semantic net is then incorporated into the ontological semantic net. We are building a natural language processing system called HAG (Hypertext Arc Generator) to automate this task.

References to concepts in the semantic net form the basis for hypertext links in SNITCH: every SNITCH hypertext link is interpreted as a path that starts at a certain point in a text node, moves across to the semantic net, traverses a sub-path through the semantic net, and then returns to some other point in the input, perhaps but not necessarily in the same node.

SNITCH is implemented in C and runs on Sun and DEC (UNIX) platforms. The primary user interface to SNITCH is a hypertext browser with a simple query language. When a user enters the SNITCH system, he or she can select a path shape from a list of alternatives, or use the default path shape. As the user visits a text node, any portion of the text that serves as the starting point of a path conforming to the current path shape is highlighted. The user can then choose to follow that path, thereby visiting some other node, or the user can change the path shape. (In the current version of SNITCH, documents only contain text.)

5.2 Maintaining Document Consistency in SNITCH

We showed above that simple hypertext documents form a category **HT**. Semantic nets also form a category **SN**, by analogy with the category of directed, labeled graphs. To preserve the consistency of SNITCH documents, then, we need to implement the appropriate representation functors between **HT** and **SN**, as discussed in Section 4.1. Specifically, this means that changes to the hypertext representation, such as the addition of a node or

deletion of a concept reference, must be reflected in the abstract document. Consistency also requires that changes to the knowledge representation language version of the semantic net must be posted to the abstract document, as well as the hypertext representation.

To maintain consistency, any structural change to the semantic net must also be reflected in the hypertext. In SNITCH this means that when new concepts are introduced to the ontological semantic net, the whole corpus needs to be scanned for references to the new concept. Augmentation of the document semantic net does not entail a structural change to the semantic net; therefore, documents can be incorporated into the system incrementally. This has the desirable effect of automatically creating new hypertext links between the new text and the existing body of text.

6 Conclusion

It is no longer appropriate to view a document as a single object with a single representation — a given document may have several concrete representations, each manifesting the meaning of the document. A characterization of the various concrete representations and the mappings between them as categories and functors, respectively, helps distinguish the essential structural information that must be preserved from the details that can be changed without altering the meaning of the document. We have considered these ideas in terms of the design of the SNITCH system, an experimental document processing system in which documents are represented as texts and as semantic nets.

Basic ideas from category theory have helped us understand the issues involved in the design of systems to process multi-representational documents. As Goguen points out [3], category theory is useful for understanding the essential nature of mathematical objects, and it helps determine the right level of abstraction for developing them. In particular, we can identify two specific insights gained by thinking about SNITCH in light of the ideas presented in this paper.

- The semantic net representation of the document should be treated as an object in its own right, with its own structure and meaning, as opposed to a simple data structure that merely describes the textual portion of the document. Without this treatment, the impact of structural changes to the semantic net would be overlooked.

- Incremental mappings between the abstract document and its various concrete representations are the key to consistency between representations.

In retrospect, neither of these insights is surprising. However, we believe that considering these issues has led to a better understanding of the SNITCH system in particular, and of document processing systems in general.

References

[1] Pehong Chen and Michael Harrison. Multiple representation document development. *Computer*, 21(1):15–31, January 1988.

[2] R. D. Tennent. *Semantics of Programming Languages.* Prentice Hall, 1991.

[3] Joseph Goguen. A categorical manifesto. *Mathematical Structures in Computer Science*, 1(1):49–67, 1991.

[4] Benjamin Pierce. A taste of category theory for computer scientists. Technical Report CMU-CS-90-113, Carnegie–Mellon University, March 1990.

[5] Michael Barr. *Category Theory for Computing Science.* Prentice Hall, 1990.

[6] Jakob Nielsen. *Hypertext & Hypermedia.* Academic Press, 1990.

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

[8] C. J. Date. *An Introduction to Database Systems.* Addison Wesley, 1990.

[9] James Mayfield and Charles Nicholas. SNITCH: Augmenting hypertext documents with a semantic net. *International Journal of Intelligent and Cooperative Information Systems*, 1994. To appear.

[10] Robert Wilensky. Some problems and proposals for knowledge representation. Memorandum UCB/CSD 87/351, University of California, Berkeley Electronic Research Laboratory, 1987.