Category Theory Approach to Fusion of Wavelet-Based Features

Scott A. DeLoach
Air Force Institute of Technology
Department of Electrical and Computer Engineering
Wright-Patterson AFB, Ohio 45433
Scott.DeLoach@afit.af.mil

Mieczyslaw M. Kokar
Northeastern University
Department of Electrical and Computer Engineering
Boston, Massachusetts 02115
kokar@coe.neu.edu

Abstract

This paper discusses the application of category theory as a unifying concept for formally developed information fusion systems. Category theory is a mathematically sound technique used to capture the commonalties and relationships between objects. This feature makes category theory a very elegant language for describing information fusion systems and the information fusion process itself. After an initial overview of category theory, the paper investigates the application of category theory to a wavelet based multisensor target recognition system, the Automatic Multisensor Feature-based Recognition System (AMFRS), which was originally developed using formal methods.

1. Introduction

The goal of information fusion is to combine multiple pieces of data in a way so we can infer more information than what is contained in the individual pieces of data alone. This requires us to be able to determine how the individual pieces of data are related. It would also be nice if we could describe this relationship between data in a formal way so that we can automatically reason over the process without the use of unreliable and brittle heuristics. In this paper we present category theory as a unifying concept for formally defining information fusion systems. The goal of category theory is to define the relationships between objects in a category of related objects. Category theory also provides operators that allow us to reason over these relationships. In previous research we have shown category theory to be useful for defining relationships between object classes in object-oriented systems [1] and now we do the same for information fusion systems.

The first section of the paper is a tutorial on algebraic specifications and category theory. Next we describe a formally defined fusion system, the Automatic Multisensor Feature-based Recognition System (AMFRS), and describe how we could incorporate category theory constructs to provide a provably correct technique for implementing the system.

2. Theories and Specifications

The notation generally used to capture the formal definitions of systems is a *formal specification*. There are two types of formal specifications commonly used to describe the behavior of software: operational and definitional. An operational specification is a "recipe" for an implementation that satisfies the requirements while a definitional specification describes behavior by listing the properties that an implementation must posses. Definitional specifications have several advantages over operational specifications because they are generally shorter and clearer than operational specifications, easier to modularize and combine, and easier to reason about, which is the key reason they are used in automated systems.

It is recognized that creating correct, understandable formal specifications is difficult, if not impossible, without the use of some structuring technique or methodology. Algebraic theories provide the advantages of definitional specifications along with the desired structuring techniques. Algebraic theories are defined in terms of collections of values called *sorts*, *operations* defined over the sorts, and *axioms* defining the semantics of the sorts and operations. The structuring of algebraic theories is provided by category theory operations and provides an elegant way in which to combine smaller algebraic theories into larger, more complex theories.

Categories are an abstract mathematical construct consisting of *category objects* and *category arrows*. In general, category objects are the objects in the category of interest while category arrows define a mapping from the internal structure of one category object to another. In our research, the category objects of interest are algebraic specifications and the category arrows are specification morphisms. In this category, *Spec*, specification morphisms map the sorts and operations of one algebraic specification into the sorts and operations of a second algebraic specification such that the axioms in the first specification become provable theorems in the second specification. Thus, in essence, a specification morphism defines an embedding of one specification into a second specification.

2.1. Algebraic Specification

In this section, we define the important aspects of algebraic specifications and how to combine them using category theory operations to create new, more complex specifications. As described above, category theory is an abstract mathematical theory used to describe the external structure of various mathematical systems. Before showing its use in relation to algebraic specifications, we give a formal definition [6].

Category. *A category C is comprised of*

- *a collection of things called C-objects;*
- *a collection of things called C-arrows;*
- *operations assigning to each C-arrow f a C-object dom f (the domain of f) and a C-object cod f (the "codomain" of f). If a = dom f and b = cod f this is displayed as*

$$f: a \to b \qquad \text{or} \qquad a \xrightarrow{f} b$$

- *an operation, "$\circ$", called composition, assigning to each pair $\langle g, f\rangle$ of C-arrows with dom g = cod f, a C-arrow $g \circ f$: dom f $\to$ cod g, the composite of f and g such that the Associative Law holds: Given the configuration*

$$a \xrightarrow{f} b \xrightarrow{g} c \xrightarrow{h} d$$

of C-objects and C-arrows, then

$$h \circ (g \circ f) \quad = \quad (h \circ g) \circ f.$$

- *an assignment to each C-object, b, a C-arrow, $id_{b:\ b\to b}$, called the identity arrow on b, such that the Identity Law holds: For any C-arrows f: a $\to$ b and g: b $\to$ c*

$$id_b \circ f = f \ \text{ and } \qquad g \circ id_b = g.$$

2.1.1. The Category of Signatures

In algebraic specifications, the structure of a specification is defined in terms of an abstract collection of values, called *sorts* and operations over those sorts. This structure is called a *signature* [7]. A signature describes the structure that an implementation must have to satisfy the associated specification; however, a signature does not specify the semantics of the specification. The semantics are added later via axiomatic definitions.

Signature. *A signature $\Sigma = \langle S, \Omega\rangle$, consists of a set S of sorts and a set Ω of operation symbols defined over S. Associated with each operation symbol is a sequence of sorts called its rank. For example, $f{:}s_1,s_2,\ldots,s_n \to s$ indicates that f is the name of an n-ary function, taking arguments of sorts s_1, s_2, …, s_n and producing a result of sort s. A nullary operation symbol, c: $\to$ s, is called a constant of sort s.*

An example of a signature is shown in Figure 1. In the signature RING there is one sort, ANY, and five operations defined on the sort.

```
signature Ring is
sorts ANY
operations
    plus   :  ANY × ANY  → ANY
    times  :  ANY × ANY  → ANY
    inv    :  ANY        → ANY
    zero   :             → ANY
    one    :             → ANY
end
```

Figure 1. Ring Signature

In our research, signatures define the required structure for formally describing wavelet-based models. Signatures provide the ability to define the internal structure of a specification; however, they do not provide a method to reason about relationships between specifications. To create a theory of information fusion using algebraic specifications, operations to define relations between specifications must be available. There must be a well-defined theory about how specifications relate to one another.

As might be expected, signatures (as the "C-objects") with the correct "C-arrows" form a category that is of great interest in our research. For signatures, the C-arrows are called *signature morphisms* [7]. Signatures and their associated signature morphisms form the category, *Sign*.

Signature Morphism. *Given two signatures $\Sigma = \langle S, \Omega\rangle$ and $\Sigma' = \langle S', \Omega'\rangle$, a signature morphism $\sigma : \Sigma \to \Sigma'$ is a pair of functions $\langle \sigma_S : S \to S', \sigma_\Omega : \Omega \to \Omega'\rangle$, mapping sorts to sorts and operations to operations such that the sort map is compatible with the ranks of the operations, i.e., for all operation symbols $f{:}s_1,s_2,\ldots,s_n \to s$ in Ω, the operation symbol $\sigma_\Omega(f){:}\sigma_S(s_1), \sigma_S(s_2),\ldots,\sigma_S(s_n) \to \sigma_S(s)$ is in Ω'. The composition of two signature morphisms, obtained by composing the functions comprising the signature morphisms, is also a signature morphism. The identity signature morphism on a signature maps each sort and each operation onto itself. Signatures and signature morphisms form a category, **Sign**, where the signatures are the C-objects and signature morphisms are the C-arrows.*

Given the signatures RING from Figure 1 and RINGINT from Figure 2, a signature morphism σ : RING $\to$ RINGINT, is shown in Figure 3. As required by the definition of a signature morphism, σ consists of two functions, σ_S and σ_Ω as shown. σ_S maps the sort ANY to Integer while σ_Ω maps each operation to an operation with a compatible rank.

Signature morphisms map sorts and operations from one signature into another and allow the restriction of sorts as well as the restriction of the domain and range of operations. However, to build up more complex signatures, introduction of new sorts and operations into a signature is required. This is accomplished via a signature *extension*.

Spec RingInt **is**
sorts Integer
operations

+	:	Integer × Integer	→ Integer
×	:	Integer × Integer	→ Integer
-	:	Integer	→ Integer
0	:		→ Integer
1	:		→ Integer

end

Figure 2. Integer Ring Signature

$$\sigma_S = \{\text{ANY} \mapsto \text{Integer}\}$$

$$\sigma_\Omega = \{\text{plus} \mapsto +, \text{times} \mapsto \times, \text{inv} \mapsto -, \text{zero} \mapsto 0, \text{one} \mapsto 1\}$$

Figure 3. Signature Morphism

Extension. *A signature $\Sigma_2 = \langle S_2, \Omega_2\rangle$ extends a signature $\Sigma_1 = \langle S_1, \Omega_1\rangle$ if $S_1 \subseteq S_2$ and $\Omega_1 \subseteq \Omega_2$.*

Signature extensions allow the definition of entirely new signatures and the growth of complex signatures from existing signatures.

2.1.2. The Category of Specifications

To model semantics, signatures are extended with *axioms* that define the intended semantics of the signature operations. A signature with associated axioms is called a *specification* [7].

Specification. *A specification SP is a pair $\langle \Sigma, \Phi\rangle$ consisting of a signature $\Sigma = \langle S, \Omega\rangle$ and a collection Φ of Σ-sentences (axioms).*

Although a specification includes semantics, it does not implement a program nor does it define an implementation. A specification only defines the semantics required of a valid implementation. In fact, for most specifications, there are a number of implementations that satisfy the specification. Implementations that satisfy all axioms of a specification are called models of the specification [7]. To formally define a model, we first define a Σ-algebra [7].

Σ-algebra or Σ-model. *Given a signature $\Sigma = \langle S, \Omega\rangle$, a Σ-algebra $A = \langle A_S, F_A\rangle$ consists of two families:*

- *a collection of sets, called the carriers of the algebra, $A_S = \{A_s \mid s \in S\}$; and*
- *a collection of total functions, $F_A = \{f_A \mid f \in \Omega\}$ such that if the rank of f is $s_1, s_2, \ldots, s_n \rightarrow s$, then f_A is a function from $A_{s1} \times A_{s2} \times \ldots \times A_{sn}$ to A_s. (The symbol × indicates the Cartesian product of sets here.)*

Model. *A model of a specification $SP = \langle \Sigma, \Phi\rangle$ is a Σ-algebra, M, such that M satisfies each Σ-sentence (axiom) in Φ. The collection of all such models M is denoted by Mod[SP]. The sub-category of **Mod(Σ)** induced by Mod[SP] is also denoted by Mod[SP].*

An example of a specification is shown in Figure 4. This specification is the original RING signature of Figure 1 enhanced with the axioms that define the semantics of the operations. Valid models of this specification include the set of all integers, *Z*, with addition and multiplication as well as the set of integers modulo 2, $Z_2 = \{0, 1\}$, with the inverse operation (-) defined to be the identity operation.

As signatures have signature morphisms, specifications also have specification morphisms. Specification morphisms are signature morphisms that ensure that the axioms in the source specification are theorems (are provable from the axioms) in the target specification. Showing that the axioms of the source specification are theorems in the target specification is a proof obligation that must be shown for each specification morphism. Specifications and specification morphisms enable the creation and modification of specifications that correspond to valid signatures within the category *Sign*. However, before we can formally define a specification morphism, we must first define a *reduct* [7].

```
spec Ring is
sorts ANY
operations
    as defined in Figure 1
axioms
    ∀a,b,c ∈ ANY
        a plus (b plus c) = (a plus b) plus c
        a plus b = b plus a
        a plus zero = a
        a plus(inv a) = zero
        a times (b times c) = (a times b) times c
        a times one = a
        one times a = a
        a times (b plus c) = (a times b) plus (a times c)
        (a plus b) times c = (a times c) plus (b times c)
end
```

Figure 4. Ring Specification

Reduct. *Given a signature morphism $\sigma: \Sigma \rightarrow \Sigma'$ and a Σ'-algebra A', the σ-reduct of A', denoted $A'|_\sigma$, is the Σ-algebra $A = \langle A_S, F_A\rangle$ defined as follows (with $\Sigma = \langle S, \Omega\rangle$):*

$$A_S = A_{\sigma(s)}' \text{ for } s \in S, \text{ and}$$
$$f_A = (\sigma(f))_{A'}, \text{ for } f \in \Omega$$

A reduct defines a new Σ-algebra (or Σ-model) from an existing Σ'-algebra. It accomplishes this by selecting a set or functions for each sort or operation in Σ based on the signature morphism from Σ to Σ '. Thus if we have a signature, Σ ', and a Σ '-model, we can create a Σ-model for a second signature, Σ, by defining a signature morphism between them and calculate the associated reduct. A reduct is now used to extend the concept of a signature morphism to form a *specification morphism* [7].

Specification Morphism. *A specification morphism from a specification SP = ⟨Σ, Φ⟩ to a specification SP' = ⟨Σ ', Φ'⟩ is a signature morphism σ: Σ → Σ ' such that for every model M ∈ Mod[SP'],* $M|_\sigma \in$ *Mod[SP]. The specification morphism is also denoted by the same symbol, σ: Σ → Σ '.*

We now turn to the definition of theories and theory presentations. Basically a *theory* is the set of all theorems that logically follow from a given set of axioms [6]. A *theory presentation* is a specification whose axioms are sufficient to prove all the theorems in a desired theory but nothing more. Put succinctly, a theory presentation is a finite representation of a possibly infinite theory. To formally define a theory and theory presentation we must first define *logical consequence* and *closure* [6].

Logical Consequence. *Given a signature Σ, a Σ-sentence φ is said to be a logical consequence of the Σ-sentences* $\varphi_1,\ldots,\varphi_n$*, written* $\varphi_1,\ldots,\varphi_n \models \varphi$*, if each Σ-algebra that satisfies the sentences* $\varphi_1, \ldots ,\varphi_n$ *also satisfies φ.*

Closure, Closed. *Given a signature Σ, the closure, closure(Φ), of a set of Σ-sentences Φ is the set of all Σ-sentences which are the logical consequence of Φ, i.e., closure(Φ) = {φ | Φ |= φ}. A set of Σ-sentences Φ is said to be closed if and only if Φ = closure(Φ).*

Theory, presentation. *A theory T is a pair ⟨Σ, closure(Φ)⟩ consisting of a signature Σ and a closed set of Σ-sentences, closure(Φ). A specification ⟨Σ, Φ⟩ is said to be a presentation for a theory ⟨Σ , closure(Φ)⟩. A model of a theory is defined just as for specifications; the collection of all models of a theory T is denoted Mod[T]. Theory morphisms are defined analogous to specification morphisms.*

Specification morphisms complete the basic tool set required for defining and refining specifications. This tool set can now be extended to allow the *combination*, or composition, of existing specifications to create new specifications. This is where category theory is extremely useful in information fusion. Often two specifications that were originally extensions from the same ancestor need to be combined. Therefore, the desired combined specification consists of the unique parts of two specifications and some "shared part" that is common to both specifications (the part defined in the shared ancestor specification). This combining operation is called a *colimit* [6]. The colimit operation creates a new specification from a set of existing specifications. This new specification has all the sorts and operations of the original set of specifications without duplicating the "shared" sorts and operators. To formally define a colimit, we must first define a cone (or *cocone*) [6].

Cone. *Given a diagram D in a category C and a C-object c, a cone from the base D to the vertex c is a collection of C-arrows* $\{f_i: d_i \to c \mid d_i \in D\}$*, one for each object* d_i *in the diagram D, such that for any arrow* $g: d_i \to d_j$ *in D, the diagram shown in Figure 5 commutes i.e.,* $g \circ f_j = f_i$.

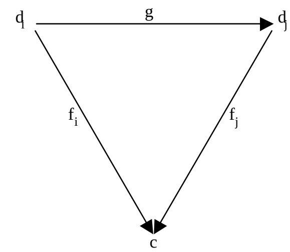

Figure 5. Cone Diagram

Colimit. *A colimit for a diagram D in a category C is a C-object c along with a cone* $\{f_i: d_i \to c \mid d_i \in D\}$ *from D to c such that for any other cone* $\{f_i': d_i \to c' \mid d_i \in D\}$ *from D to a vertex c', there is a unique C-arrow f: c → c' such that for every object* d_i *in D, the diagram shown in Figure 6 commutes (i.e.,* $f \circ f_i = f_i'$*).*

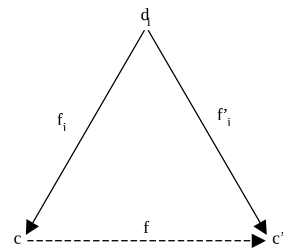

Figure 6. Colimit Diagram

Conceptually, the colimit of a set of specifications is the "shared union" of those specifications based on the morphisms between the specifications. These morphisms define equivalence classes of sorts and operations. For example, if a morphism for specification A to specification B maps sort α to sort β, then α and β are in the same equivalence class and thus is a single sort in the colimit specification of A, B, and the morphism between them. Therefore, the colimit operation creates a new specification, the colimit specification, and a cone morphism from each specification to the colimit specification. These cone morphisms satisfy the condition that the translation of any sort or operation along any of the morphisms in the diagram leading to the colimit specification is equivalent. An example of the colimit operation is shown in Figure 7 and Figure 8. Given the BIN-REL, REFLEXIVE, and TRANSITIVE specifications in Figure 7, the "colimit specification" would be the PRE-ORDER specification as shown in the diagram in Figure 8. Notice that the sorts E, X, and T belong to the same equivalence class in PRE-ORDER. Likewise, the operations •, =, and < also form an equivalence class in PRE-ORDER. Thus PRE-ORDER defines a specification with one sort, denoted by {E, X, T} and one operation, denoted by {•, =, <}, which is both transitive and reflexive. The specification BIN-REL defines the "shared" parts of the colimit but adds nothing to the final specification.

```
spec  Bin-Rel  is
sorts  E
operations
      •    :  E, E    → Boolean
end

spec  Reflexive  is
sorts  X
operations
      =    :  X, X   → Boolean
axioms
      ∀ x ∈ X   x = x
end

spec  Transitive  is
sorts  T
operations
      <    :  T, T    → Boolean
axioms
      ∀ x, y, z ∈ T    (x < y ∧  y < z) ⇒ x < z
end

spec  Pre-Order  is
sorts  {E, X, T}
operations
      {•, =, <} : {E, X, T}, {E, X, T} → Boolean
axioms
      ∀ x, y, z ∈  {E, X, T}
            x  {•, =, <}  x
            (x  {•, =, <}  y  ∧  y  {•, =, <} z) ⇒ x  {•, =, <}  z
end
```

Figure 7. Specification Colimit Example

A category in which the colimit of all possible C-objects and C-arrows exists is called *cocomplete*. As shown by Goguen and Burstall [2], the category *Sign* and *Spec* are both cocomplete; therefore, the colimit operation may be used freely within the category *Spec* to define the construction of complex theories from a group of simpler theories.

Using morphisms, extensions, and colimits as a basic tool set, there are a number of ways that specifications can be constructed [7]:

1. Build a specification from a signature and a set of axioms;
2. Form the union of a collection of specifications;
3. Translate a specification via a signature morphism;
4. Hide some details of a specification while preserving its models;
5. Constrain the models of a specification to be minimal;
6. Parameterize a specification; and
7. Implement a specification using features provided by others.

Many of these methods are useful in specifying and implementing information fusion systems. For instance, if we can define the shared part of two types of data, we can formally combine them using a colimit.

2.2. Functors

The previous sections defined the basic categories and construction techniques used to build large-scale software specifications. In this section, we extend these concepts further to define models of specifications and how they are related to the construction techniques used to create their specifications. Before describing this relationship, we define the concept of a *functor* that maps C-objects and C-arrows from one category to another in such a way that the identity and composition properties are preserved [5].

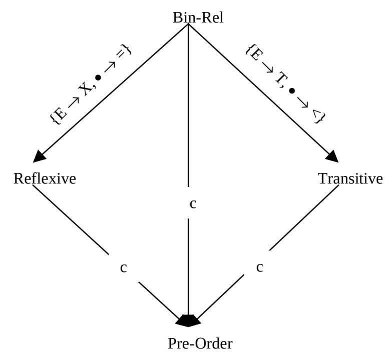

Figure 8. Example Colimit Diagram

Functor. *Given two categories **A** and **B**, a functor F: **A** → **B** is a pair of functions, an object function and a mapping function. The object function assigns to each object X of category **A** an object F(X) of **B**; the mapping function assigns to each arrow f: X → Y of category **A** an arrow F(f) : F(X) → F(Y) of category **B**. These functions satisfy the two requirements:*

$F(1_X) = 1_{F(X)}$
for each identity 1_x *of* $\mathbf{A}$

$F(g \circ f) = F(g) \circ F(f)$
for each composite $g \circ f$ *defined in* $\mathbf{A}$

Basically a functor is a morphism of categories. Actually, we have already presented two functors: the *reduct* functor that maps models of one specification (in the category $Mod[X_1]$) into models of a second specification (in the category $Mod[X_2]$) and the *models* functor that maps specifications in the category *Spec* to their category of models, $Mod[X]$, in *Cat*, the category of all sufficiently small categories.

3. AMFRS

To show applicability of the category theoretic notions described above to information fusion systems, we will discuss a case study of Automatic Multisensor Feature-based Recognition System (AMFRS) [4], which was originally developed using a model-based approach. In this case study, we transform the AMFRS framework into an equivalent system using a category theoretic approach. First we will discuss the original system and then show its equivalent structure using algebraic specifications and category theory.

3.1. Model-Theory Based Framework

In the original model-based development approach, wavelet-based models were developed for integration into the AMFRS to help recognize targets. AMFRS uses a model-based framework to describe how to combine information contained in the wavelets for use in the system. Within this framework, models were developed to help recognize targets based on wavelet coefficients that could be interpreted as meaningful features of the target.

In this framework, models were developed based on a *language* and its associated *theory* that described the semantics of the language. To combine languages and theories, three operators are used: reduction, expansion, and union. In general, the *reduction* operator removes symbols from a language along with all the sentences in which it exists in its associated theory. Expansion is the opposite. *Expansion* allows us to add symbols and new sentences about those symbols to the language. Finally, the *union* operator combines the symbols and sentences from two different language/theory pairs into a single language and a single theory.

Using these operators, Korona created a framework for combining languages and theories about two different types of sensor data into a single fused language and theory. This framework is shown in Figure 9. In Figure 9, we show only the language composition process. The theory fusion process is identical. In this example, we assume there are two sensors whose data is described by two languages L_r and L_i. These languages are extended to the languages L_r^e and L_i^e by adding symbols denoting operations on a subset of the wavelet coefficients used to describe the sensor data. These subsets of coefficients represent those coefficients that will be part of the final fused language. The coefficients are selected by the designer based on knowledge of the wavelet coefficients and their relationship to features in targets of interest.

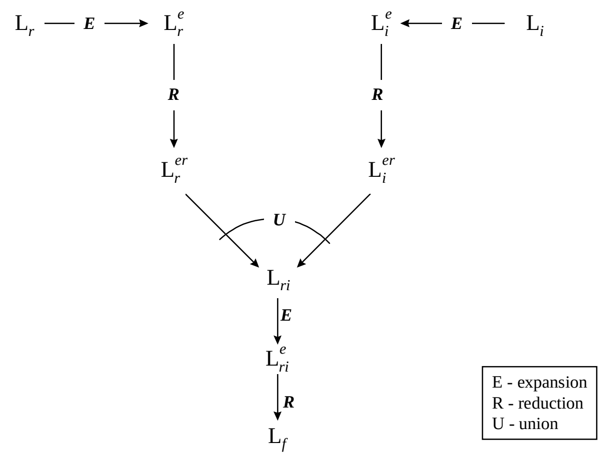

Figure 9. Model-Theory Based Framework

After the necessary symbols have been added to the languages, L_r^e and L_i^e are reduced by removing all the symbols not related to the coefficients selected for use in the final fused language. The new reduced languages, L_r^{er} and L_i^{er}, are then combined into a single language, L_{ri}, by the union operation. This language contains all the symbols representing the coefficients and operations on them required to construct the final fused language.

The last two steps in the process create our final fused language, L_f. First, L_{ri} is extended to L_{ri}^e by adding symbols denoting operations that combine the coefficients from L_r^{er} and L_i^{er}. Then, we create L_f by removing the symbols denoting those operations that do not work on the fused set of coefficients.

3.2. An Equivalent Categoric Framework

Before we convert the AMFRS model-based framework into a categoric framework, a few observations are necessary. First, the language and theory combination used in AMFRS is basically equivalent to an algebraic specification. An algebraic specification defines a set of sorts, operations over those sorts, and axioms that define the semantics of the operations. Constants, relations and functions defined via language symbols are defined as *operations* in an algebraic specification. Sentences of a theory translate to *axioms* in an algebraic specification. Algebraic *sorts* define a collection of values used in the operations.

The model-based expansion, reduction, and union operators also have counterparts in category theory. The basic operator in category theory is the *morphism*. In the category of *Spec*, which includes all possible algebraic specifications, these morphisms are *specification morphisms* that define how one specification is embedded in a second specification. That is, it defines a mapping from the sorts and operations of the first specification into the sorts and operations of the second specification in such a way as to ensure the axioms of the first specification are theorems of the second specification (i.e., the axioms hold in the second specification under the defined mapping of sorts and operations). Thus a specification morphism can be used to define an expansion as well as a reduction (they are basically inverses of each other). If we have an expansion of specification *A* into specification *B*, in effect we have a morphism from *A* to *B*. Likewise, a reduction of specification *A* to specification *B*, indicates morphism from *B* to *A*. The language union operator can also be modeled easily using the category theory *colimit* operation. The colimit operation combines two (or more) specifications, automatically creating a morphism between the original specifications and the resulting colimit specification. If two specifications being combined using a colimit operation share common parts (e.g., they both use integers), these parts can be specified as common by defining morphisms from the common, or shared, specification to the individual specifications. This shared specification, along with the associated morphisms, are included in the colimit operation. The result of this is that the shared parts of the two specifications are not duplicated.

The conversion of the model-based framework into a category theoretic framework is shown in Figure 10. In this framework, the languages and their associated theories are converted to algebraic specifications (or *theory presentations*) and reductions and extensions are converted to morphisms. Note that a reduction from *A* to *B* results in a morphism from *B* to *A*. The union operation is converted to a colimit operation. The *S* specification denotes any shared part of specifications T_r^{er} and T_i^{er}. In this case it might include domain information about wavelets, targets, etc.

Figure 11 represents a simplification of the category theoretic setting shown in Figure 10. Basically, the morphisms σ_3, σ_4, and σ_8 from Figure 10 have been combined into morphism σ_{15} of Figure 11. This is possible since all the sorts, operations, and axioms removed by σ_3 and σ_4 can be carried along without changing the semantics. As we see when we get to the model creation phase, carrying along these extra sorts, operations, and axioms is an advantage.

Figure 12 is an even further simplification of the category theoretic setting of Figure 10. In Figure 12, the morphisms σ_1, σ_2 and σ_7 from Figure 10 have been combined into morphism σ_{14}. In this framework, we combine the two basic specifications together via the colimit operation *before* we insert any knowledge about which wavelet coefficients correspond to which interpretable features.

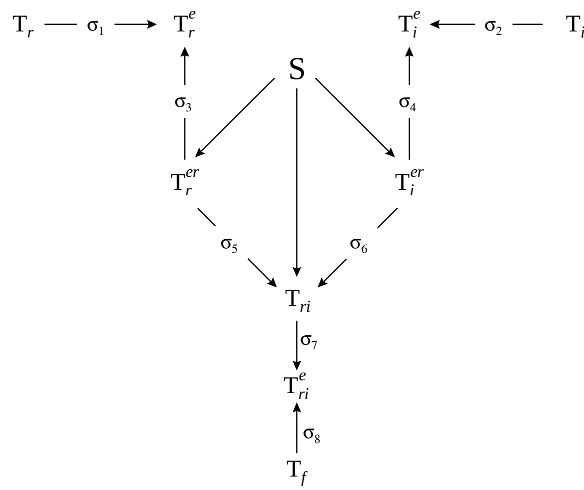

Figure 10. Categorical Framework

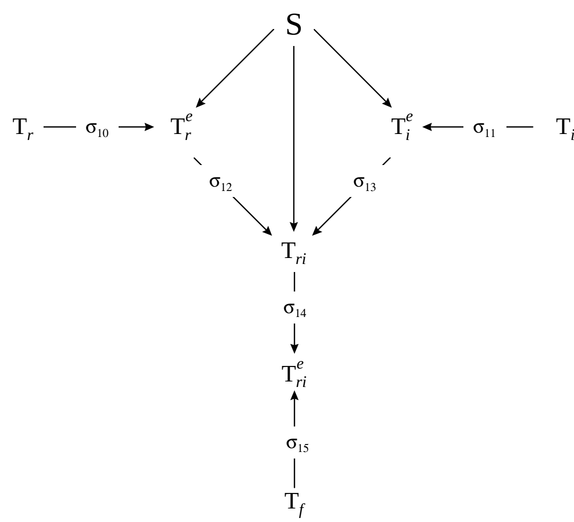

Figure 11. Simplified Categorical Setting

Since all the operations used to expand the basic specifications have a well defined interpretation in the expanded specifications (cf. [4]), the morphism σ_{14} becomes a *definitional extension* and the subdiagram contained in the dotted box becomes an *interpretation*. An interpretation basically says that we can build a model of T_f from a model of T_{ri}. This is a powerful construct in category theoretic software development tools such as Specware [3].

Finally Figure 13 describes how we create models in our category theoretic framework. In Figure 13, *MOD* represents the *model functor*, which takes specifications from the category *Spec* and maps them to a valid category of models, denoted *MOD[Spec]*, in the category *Cat* (the category of all sufficiently small categories). The nice part about the category theoretic framework we have come up with is that each morphism, σ: $A \rightarrow B$, induces a *reduct functor*, $|_\sigma$, that automatically maps models of B to models of *A*. Therefore if we create a valid model for *B,* we automatically get a valid model for *A*! Following the flows of reduct functors in Figure 13, we now see that if we can create a valid model of T_f-*as*-T_{ri} (M_{ri}^e as pointed at by the large arrow in Figure 13) we can automatically create the valid models M_r, M_i, M_{ri}, and M_f from T_r, T_i, T_{ri}, and T_f respectively. Not only are these models consistent with their individual theories, but since all the models are based on a single initial model, they are consistent with each other as well.

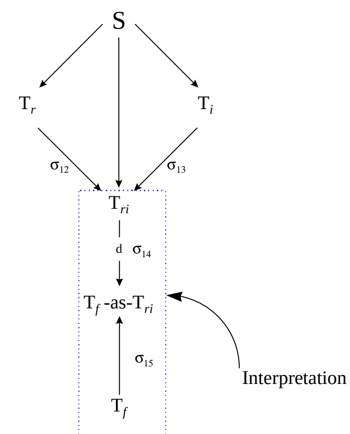

Figure 12. Theory Interpretation

4. Implications

There are many positive implications of putting the AMFRS design into a category theoretic setting. First, there is no information loss in translating languages and theories into algebraic specifications. In fact, we gain modeling ability by adding the notion of a sort. By using sorts, we can precisely define operation signatures. Also, the notions of morphisms, definitional extensions, colimits, and interpretations give us a wide variety of tools with well-defined meanings. We can prove when morphisms and definitional extensions exist as well as *construct* the resulting colimit specification based on a set of specifications and morphisms. All in all, category theory provides us a much greater capability to prove *relationships* between specifications. Finally, the categorical setting allows us to construct, in a provably correct manner, consistent sets of models required by the AMFRS system. All we have to do is construct one specific model and the models required by AMFRS can be generated automatically. The bottom line is, you lose nothing and gain a lot by using category theory in the development of formal information fusion systems such as AMFRS.

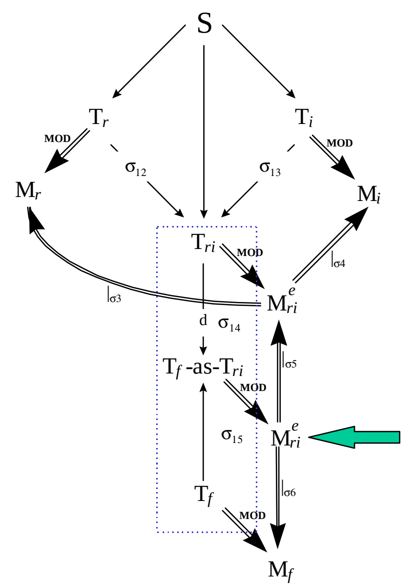

Figure 13. Model Creation using Theory Interpretation

5. References

1. DeLoach, Scott A. and Thomas C. Hartrum. "A Theory-Based Representation for Object-Oriented Domain Models," to appear in *IEEE Transactions on Software Engineering*.

2. Goguen, J. A. and R. M. Burstall. "Some Fundamental Algebraic Tools for the Semantics of Computation Part I: Comma Categories, Colimits, Signatures and Theories," *Theoretical Computer Science*, 31:175-209 (1984).

3. Jullig, Richard and Yellamraju V. Srinivas. "Diagrams for Software Synthesis." *Proceedings of the Knowledge Based Software Engineering Conference*. IEEE 1993.

4. Korona, Z. *Model-Theory Based Feature Selection for Multisensor Recognition*. Ph.D. Thesis, Northeastern University, 1996.

5. MacLane, Saunders and Birkhoff. *Algebra.* New York, NY: Chelsea Publishing Company, 1993.

6. Srinivas, Yellamraju V. *Category Theory Definitions and Examples.* Technical Report, Department of Information and Computer Science, University of California, Irvine, February 1990. TR 90-14.

7. Srinivas, Yellamraju V. *Algebraic Specification: Syntax, Semantics, Structure.* Technical Report, Department of Information and Computer Science, University of California, Irvine, June 1990. TR 90-15.