MATHEMATICAL FRAMEWORK FOR GENERALIZATION AND INSTANTIATION OF KNOWLEDGE

Marek Reformat
thinkS^2: thinking Software and System laboratory
Electrical and Computer Engineering, University of Alberta
ECERF Building W6-023, 9107-116 Street, Edmonton, Alberta, Canada, T6G 2V4

Keywords: Knowledge representation, protoforms, instances, category theory, fibration, abstract data types.

Abstract: Templates, patterns, and blueprints are constructs that humans use to represent highly abstract knowledge. Quality of such processes as reasoning, speaking, running, and driving depends on people's abilities to process these constructs. Recently, they have been named protoforms. On the other hand, concrete pieces of knowledge can be seen as instances of the protoforms.
A very important task is to find mechanisms that will be able to organize and control protoforms and their instances. They would provide methods for defining properties of protoforms and their instances, describing their interactions, and controling ways how they can be merged. The paper describes a concept of applying category theory to describe protoforms and their instances in a more formal way.

1 INTRODUCTION

The fundamental goal of knowledge representation is to prepare an environment suitable for performing analysis and processing of represented knowledge. Zadeh's (Zadeh, 2002) observation is that humans use patterns representing different aspects of everyday activities, such as building sentences, and making decisions. To generalize this observation, Zadeh introduced the concept of *protoforms* (prototypical forms). They represent a wide range of concepts, procedures, and schemes related to different activities.

At the time when protoforms represent a generalized knowledge, a specific pieces of knowledge are instances of protoforms. A single protoform together with its instances can be related to other protoform and their instances. In general we can talk about a network of "relations" among different protoforms and instances.

Category theory (Barr and Wells, 1999) is a discipline of mathematics dedicated to the theory of structures – it deals with structures and relationships between them in an abstract way. The basic elements of category theory are *objects* and *morphisms* (*arrows*). Category theory focuses on relations that exist among objects, it tries to describe objects via their interactions with an environment and among themselves.

In the paper, a mathematical scheme for analysis and inference about protoforms and their instances is described. The categories **Protoforms** and **Instances** are defined. Each object of the category **Protoforms** is a single protoform represented as an algebraic signature containing a set of types, and a set of operations over those types. In this category, morphisms are "relations" that translate the vocabulary (types) and operations of one signature (protoform) into the vocabulary and operations of another. Objects of the category **Instances** are instances (individuals) build on protoforms. Relations among objects of both categories, i.e., **Protoforms** and **Instances**, are defined by special relations called *fibrations*. This means that instances are fibers built over protoforms.

2 PROTOFORMS

The concept of protoforms, introduced by Zadeh in (Zadeh, 2002), represents an interesting idea of generalization. A protoform – a short of *prototype form* – is defined as "an abstracted summary". The protoform is a symbolic expression defining a construct - a concept, proposition, command, question, scenario, case,

Reformat M. (2007).
MATHEMATICAL FRAMEWORK FOR GENERALIZATION AND INSTANTIATION OF KNOWLEDGE.
In *Proceedings of the Second International Conference on Software and Data Technologies - PL/DPS/KE/WsMUSE*, pages 249-254
DOI: 10.5220/0001344802490254

or a system of such constructs. It has been already shown that protoforms have a universal character, and are useful for building intelligent systems (Kacprzyk and Zadrozny, 2005) (Yager, 2006).

Example: One of the simplest protoforms are concepts. A concept *car* is such an example. This protoform identifies components of a car, and "relations" between them. There are many concepts related to *car*, for example *racing car, sport utility car*, and all of them constitute a network of concepts.

Example: Another portoform can be a single proposition *V is B*, where *V* is a variable, and *B* is a subset indicating the allowable values for the variable. This protoform can be a building block of more complex protoforms (Yager, 2006).

Example: Protoforms can be also used to represent database query summaries (Kacprzyk and Zadrozny, 2005). In this case, a query summary such as *Most records meeting conditions B match query S* can be represented by a profotorm: *Most BRs are S*, where R means records, B is a filter, and S is a query. Evidently, as protoforms may form a hierarchy, higher level (more abstract) protoforms can be defined, for instance replacing most by a general linguistic quantifier Q: *QBRs are S*.

3 CATEGORY THEORY

3.1 Basics of Category Theory

Category theory (Barr and Wells, 1999) is a branch of mathematics that deals with structures and relationships among them. The structures are called *objects* and a relationship between two objects is called a *morphism*. The essence of category theory, as stated in (Fiadeiro, 2005), is that category theory characterizes objects in terms of their "social life". This social life represents interaction of objects among themselves and their universe (environment).

Due to the space limitation we do not provide definitions of basic concepts of category theory, such as category, morphism, functor, and universal constructions (for example, pushouts). Definitions of these concepts can be found in any category theory book, for example (Jacobs, 2001).

3.2 Fibrations

Special types of functors, that define relationships that exists among objects that belong to two different categories, are called *fibrations* (Barr and Wells, 1999). A fibration is designed to capture collections

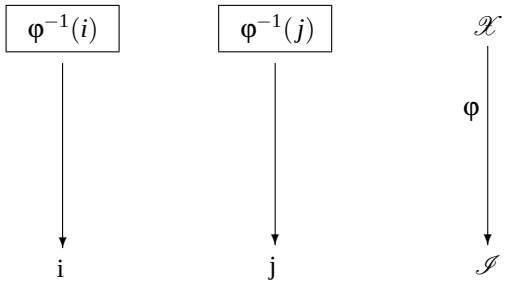

Figure 1: Fibers over *i* and *j* (each box represents a set of objects $\mathscr{X}$ that are mapped into *i* and *j* respectively.

of categories varying over a base category. An example can be collections of sets $(X_i)_{i\in I}$ varying over a base, or index, set *I*, Fig. 1. Let's consider a functor $\varphi : \mathscr{X} \to \mathscr{I}$. The sets in the category $\mathscr{X}$ appear as **fibers** over elements/objects of category $\mathscr{I}$

$$\varphi^{-1}(i) = \{x \in X | \varphi(x) = i\}$$

for each $i \in \mathscr{I}$. In other words, a fiber is a collection of items of one category that can be mapped (via fibration) into a single element (object) of another category.

A formal definition of fibration is based on the concept of cartesian morphisms [1].

Definition 1 *Let $P : \mathscr{E} \to \mathscr{C}$ be a functor between categories, let $f : C \to D$ be an arrow of $\mathscr{C}$, and let $P(Y) = D$. An arrow $u : X \to Y$ of $\mathscr{E}$ is* **cartesian** *for f and Y if (see Fig. 2 for a graphical representation):*

- $P(u) = f$
- *for any arrow $v : Z \to Y$ of $\mathscr{E}$ and any arrow $h : P(Z) \to C$ of $\mathscr{C}$ for which $f \circ h = P(v)$, there is a unique $w : Z \to X$ in $\mathscr{E}$ for which $u \circ w = v$ and $P(w) = h$.*

Definition 2 *A functor $P : \mathscr{E} \to \mathscr{C}$ is a* **fibration** *if there is a cartesian arrow for every $f : C \to D$ in $\mathscr{C}$ and every object Y in $\mathscr{E}$ for which $P(Y) = D$.*

If $P : \mathscr{E} \to \mathscr{C}$ is a fibration, one also says that $\mathscr{E}$ is **fibered over** $\mathscr{C}$. In that case, $\mathscr{C}$ is the **base category** and $\mathscr{E}$ is the **total category** of the fibration. Some authors represent a fibration vertically:

$$\begin{array}{c} \mathscr{E} \\ \downarrow \\ \mathscr{C} \end{array}$$

This way of representing a fibration is very intuitive.

[1] There is also a dual concept to fibration called cofibration or opfibration. We skip the formal definition of opfibration, but elements of opcartesian (dual to cartesian) morphism are used in Section 7 to illustrate benefits of the approach we proposed in the paper.

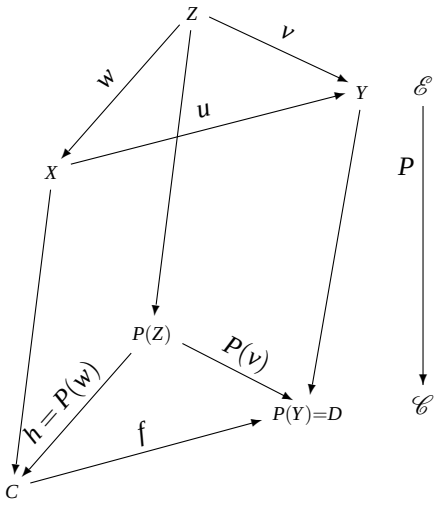

Figure 2: Formal definition of a fibration (objects *X, Y, Z* are of $\mathscr{E}$, and *C, P(Z), P(Y)=D* of $\mathscr{C}$).

4 ALGEBRAIC REPRESENTATION

4.1 Abstract Data Types and Signatures

An abstract data type (ADT) is a mathematical specification of data and operations that can be performed on the data. An ADT is created by identifying types of data that become its elements, and providing information about operations that can be performed on these identified types of data. The only information is a name of operation, its domain and co-domain. The actual implementation, i.e., values of the identified types and the results of operations on these values are not defined.

The technique of algebraic specification of abstract data types can be introduced informally by defining the concept of signature, called also an algebraic signature. The signature defines types of data that are used by the abstract data type, and a set of operations. Its formal definition is shown below.

Definition 3 *A* **signature** $SIG = (T, OP)$ *consists*

- *T, the set of types (also called sorts);*
- *OP, the set of constant and operation symbols;* K_t *is a set of constant symbols of types* $t \in T$, $OP_{w,t}$ *is a set of operation symbols with argument types* $w \in T^*$ *(*T^* *is a set of strings built using types* $t \in T$*, i.e.,* $\langle t_1, ..., t_n \rangle$*), and a range type* $t \in T$ *(*$t_{n+1} \in T$*).*

An intuitive data type: boolean can be defined using ADT. Its signature is the following:

bool =
sort bool
opns: TRUE, FALSE → bool
NOT: bool → bool
AND: bool bool → bool

engine =
sort piston, valve, engine_block, engine
opns eASSEMBLY: piston × valve
× engine_block, → engine

Figure 3: Signature of a concept *engine*.

where $T = \{bool\}$, and $K_t = \{TRUE, FALSE\}$, $OP_{w,t} = \{NOT, AND\}$. As it can be seen, there is no information about the values boolean can take, as well as results of operations *AND* and *NOT*.

4.2 Protoforms as Algebraic Signatures

High level of abstraction and universal character of both protoforms and ADT have led us to the idea of applying algebraic signatures for representing protoforms. The very aspects of ADT, such as focus on generic nature of relationships among data components and an implementation independence, make this idea very attractive.

Let us take a look at a simple example of a protoform signature. The protoform signature presented in Fig. 3 is the signature of a simple concept *engine*. It contains a number of components and a single function identifying a relation between them.

The signature *engine* can be treated as an elementary protoform, and other protoforms can be built based on it. For example, it can be extended by introducing new types and operations. The extended signature is shown in Fig. 4.

The signature *engine* can also be used as a part of more complex signatures. An example of such signature is presented in Fig. 5. This signature named *car* contains additional types and operations.

turboEngine =
sort piston, valve, engine_block, engine,
turbocharger, turbo_engine
opns eASSEMBLY: piston × valve
× engine_block, → engine
tASSEMBLY: engine × turbocharger
→ turbo_engine

Figure 4: Signature of a concept *turbo_engine*.

car =
sort engine, body, interior, wheel, car
opns cASSEMBLY: engine × body
× interior × wheel → car

Figure 5: Signature of a concept *car*.

5 PROTOFORMS AND CATEGORY THEORY

5.1 Concept

Representation of protoforms using algebraic specifications (signatures), presented in the Section 4.2, creates a possibility of formalizing protoforms and processes of their construction. Therefore, we define a category **Protoforms** in the following way.

Definition 4 *The category* **Protoforms** *has*
objects
protoforms represented by signatures
morphisms
signature morphisms – it maps the sorts and operations from one signature to another (possible morphisms are substitutions, inclusions, extensions).

Construction of category **Protoforms** provides several advantages: a more mathematical approach to dealing with protoforms, a better understanding of relations between protoforms, a systematic approach to constructing more complex protoforms.

The category **Protoforms** is presented here in the form of examples of their objects, and an example of application of the *pushout* construction.

5.2 Protoforms as Objects

The signatures presented in Fig. 3, 4, and 5 are objects of the category **Protoform**. It can be shown that there is a very simple morphism between two of them, Fig. 6. This morphism is just an extension of the sort *engine* into *turbo_engine*. The protoform *turboEngine* contains an additional type *turbocharger* and one more operation *tASSAMBLY* that represents a process of adding turbocharger to an engine.

For the purpose of the next subsection related to construction of more complex protoforms, these protoforms are redefined, Fig. 7, and the signature *car* is renamed *protoformCar*.

5.3 Protoform Category and Universal Constructs

Universal constructions can be used to define relationships among protoforms.

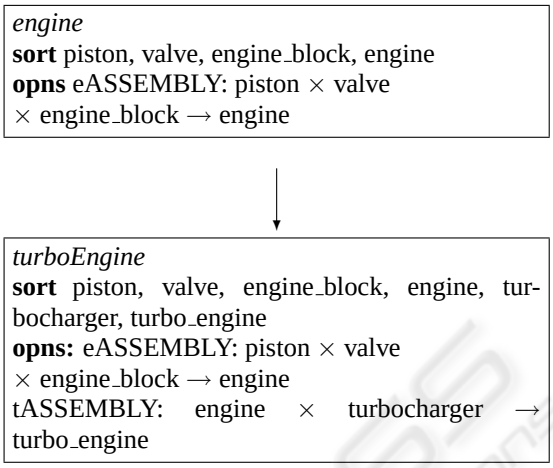

Figure 6: An example of a morphism in the category **Protoforms**.

protoformEngine =
sort *engine*
protoformTurboEngine =
sort *turboEngine*

Figure 7: Objects of **Protoform.** category

Example A pushout that uses *protoformTurboEngine* and *protoformCar* to create a new protoform - *protoformTurboCar* is shown in Fig. 8. The process of constructing a new protoform is "defined" by the mappings of the sort *engine* of *protoformEngine* to the sort *engine* of *protoformCar*, and to the sort *turboEngine* of *protoformTurboEngine*. The *engine* and *turboEngine* are "amalgamated", and the resulting protoform has the sort *turboEngine*.

6 INSTANTIATION AND CATEGORY THEORY

6.1 Models

The definition of signature (Section 4.1) identifies sorts (types) and operations, but does not provide any details about them, i.e., values and relations among these values. However, each signature can be used to create a model (an algebra). This is done by assigning concrete values to each type and operation.

Definition 5 *A model or algebra for a signature SIG consists of a T-indexed collection* $(A_t)_{t \in T}$ *of carrier sets together with a collection of suitably typed operations: each operation symbol* $OP : t_1, ..., t_n \rightarrow t_{n+1}$

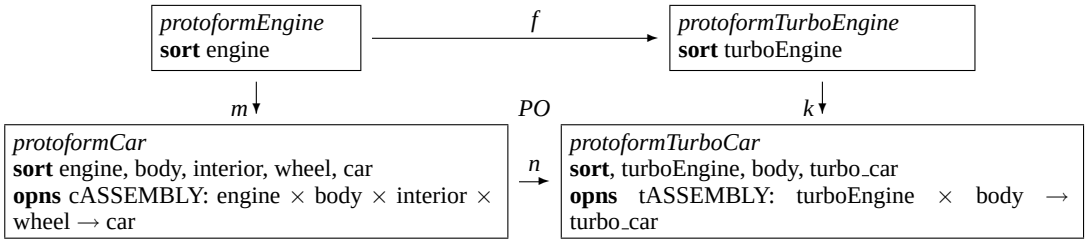

Figure 8: An example of a pushout (PO) in the category **Protoforms**.

in SIG is mapped to an actual operation $[\![OP]\!] : A_{t_1} \times \ldots \times A_{t_n} \rightarrow A_{t_{n+1}}$ *between the corresponding carrier sets. Thus a model consists of a pair* $((A_t)_{t\in T}, [\![\_]\!])$.

In other words a model represents a "concrete" piece of knowledge built based on a signature. A single signature can be used to create several models.

Example. A simple model (each type has only one possible value) that can be built based on *engine* is presented below.

$\mathbf{A_{engine}} =$
A_{piston}: $\{3-ring\_pistons\}$
A_{valves}: $\{intake\_and\_exhaust\_valves\}$
A_{engine_block}: $\{block\_without\_sparks\}$
A_{engine}: $\{diesel\_engine\}$

$[\![\mathbf{eASSEMBLY_A}]\!]$: $A_{piston} \times A_{valves} \times A_{engine\_block} \rightarrow A_{engine}$

$(3-ring\_pistons,$
$intake\_and\_exhaust\_valves,$
$block\_without\_sparks) \rightarrow diesel\_engine$

From the perspective of category theory models built based on signatures can be treated a category.

Definition 6 *The category* **Sig-Model** *of models of signatures has*
objects
$(SIG, (A_s, [\![\_]\!])$ *where* $(A_s, [\![\_]\!])$ *is a model for SIG*
morphisms
$(\phi, (H_s)) : (SIG, (A_s, [\![\_]\!]) \rightarrow (SIG', (A'_s, [\![\_]\!]')$
where each morphism consists of a morphism of signature $\phi : SIG \rightarrow SIG'$*, and a* $|SIG|$*-indexed collection of operations*

A relationship between signatures and models resembles the relationship between protoforms and instances. In order to utilize this relationship, we need to build a category of instances.

Definition 7 *The category* **Instances** *has*
objects
models of protoform signatures
morphisms
model morphisms containing signature morphims

6.2 Instantiation and Fibers

The concept of a fibration (Section 3.2) represents a very interesting way of formalizing relations between two categories. It has been found (Jacobs, 2001) that the relationship between a set of models and a single signature is a fibration. So the functor between the category of models **Sig_Model** and the category of signatures **Sig** is a fibration. This can be represented in the following way:

Sig-Model
↓
Sig

In other words the functor **Sig_Model** → **Sig** sends models to its underlying signature. A fiber over an object (signature) $SIG \in$ **Sig** is a model of the signature *SIG*. We can induce that similar relation exists between objects of the categories **Instances** and **Protoforms**.

Instances
↓
Protoforms

This defines and formalizes relationships that exist between protoforms and instances. All these observations can be summed-up into a single statement that instances are fibers over protoforms.

7 DISCUSSION

The idea of treating protoforms as a base category for instances brings a very elegant way of identifying relationships that exist among protoforms and instances. Fig. 9 represents a very simple knowledge system that will be used to explain benefits of the idea we propose in the paper. The system contains three protoforms *engine*, *turbo_engine*, and *car_turbo*. Each of the protoform signatures is a base for a set of models – instances. Instances of *engine* are represented by small circles with two explicit instances

V8 and *V6*. Similar situation is seen for *turbo_engine* – squares and *turboV8*, and *car_turbo* – triangles and *Porsche911Turbo*. There are also morphisms *f, g, k* between protoforms, and morphisms , *f', g', k', f", g"* between instances.

Fig. 9 illustrates what it means that instances are fibers over protoforms, and how this fact influences relationships between protoforms and instances. A few interpretations are included here. The first one is based on the definition of opfibration[2], and the others on its consequences:

- let us assume that the morphism *f* exists and that the system receives a piece of information *V8* which is identified as the instance of the *engine*; this means that there is a morphism *f'* such that there should be two morphisms *g'* and *k* inducing a unique morphism *k'*;
- if a new piece of knowledge *V6* appears, and it is an instance of the protoform *engine*, then the system should search for morphisms between this instance and other instances, this search is induced by the existence of morphisms between *engine* (*f*) and *turbo_engine* (*g*), as well as between *engine* and *car_turbo*; the system will constantly search for these morphisms (*f"* and *g"* in Fig. 9) until all relations identified in the definition of opfibration are satisfied;
- if a morphism (connection) is found between two instances that belong to the models of different protoforms not related to each other, the system would treat this morphism as a "hint" that both protoforms should connected; as above, the system will search for such a morphism until all relations identified in the definition of opfibration are satisfied, and the correctness of this new morphism is validated.

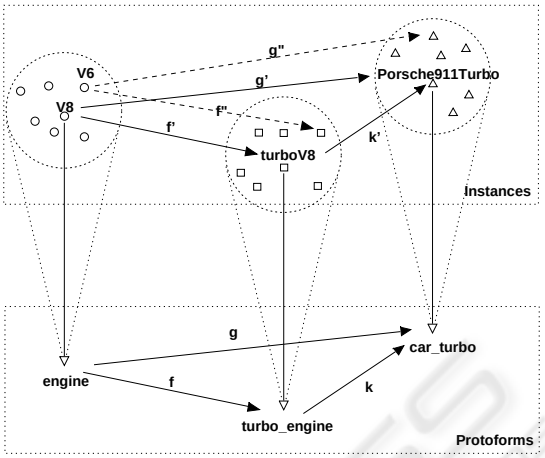

Figure 9: An example of knowledge system built based on the categories **Protoforms** and **Instances** ("balloons" represent fibers over protoform signatures).

8 CONCLUSIONS

This paper presents preliminary results of the work on application of the ADT and category theory to the concepts of protoforms and instances. It has been shown that such application is feasible and promises a number of benefits. The most important benefits include a systematic approach to dealing with protoforms and their instances; a formal way of merging protoforms and building new ones; and a formal way of validating correctness of newly established connections between protoforms and instances.

Additional work is needed towards a full utilization of such concepts as specification of protoforms, and application of Specware[3] (Kestrel_Institute, 2004), for development of protoform-based systems.

REFERENCES

Barr, M. and Wells, C. (1999). *Category Theory for Computing Science*. Les Publications CRM, Montreal.

Fiadeiro, J. L. (2005). *Categories for Software Engineering*. Springer-Verlag, Berlin.

Jacobs, B. (2001). *Categorical Logic and Type Theory*. Elsevier, Amsterdam.

Kacprzyk, J. and Zadrozny, S. (2005). Linguistic database summaries and their protoforms: towards natural language based knowledge discovery tools. In *Information Sciences*, volume 173, pages 281–304.

Kestrel_Institute (2004). *Specware 4.1 User's Manual*. Kestrel Development Corp., Palo Alto, California.

Yager, R. R. (2006). Knowledge trees and protoforms in question answering systems. In *Journal of the American Society for Information Sciences*.

Zadeh, L. A. (2002). A prototype-centered approach to adding deduction capabilities to search engines - the concept of a protoform. In *BISC Seminar*, University of California, Berkeley.

[2]Fig. 9 can be used to define opcartesian morphism required for definition of opfibration. As it was stated in Section 3.2, this definition is a dual to fibration. For details please look (Barr and Wells, 1999).

[3]An automated software development system based on principles of ADT and category theory.