Categories in Modelling

Milan Urbášek*
urbasek@dcse.fee.vutbr.cz

Abstract: The aim of this paper is to discuss possible use of category theory as a helpful formalism for certain problems involved in the area of modelling and simulation. The paper discusses general notions of category theory and shows examples of categories that can be found in the context of modelling, especially categories of models of programming languages and simulation categories of state-oriented models.

Key Words: Category theory, modelling, simulation language, bisimulation, Petri nets

1 Introduction

Category theory was introduced in 1945 as a formal mathematical tool in the algebraic topology. A category is a very flexible structure, very often referred as an algebra of functions. Due to the flexibility of a category, category theory propagated to many other areas of mathematics as for example logic and algebraic geometry. The structure of a category has been also found useful in the computer science, and the formalism of the category theory was accepted as a formal background for many ideas and concepts by computer scientists. The category theory still influences researchers trying to develop formal verification and simulation techniques and also people working on modelling and formal analysis of models.

2 Category Theory - Foundations

Before considering categories and categorical models it is necessary to introduce basic terminology from category theory. A category is, in general, more complex structure than other algebraic objects (e.g. monoids, groups, etc.). Its definition is not trivial but on the other hand, it gives a complex tool which is suitable for many purposes.

2.1 A Category

Definition: A *category* **A** consists of a set of *objects* (called *obj*(**A**)) and the set of *morphisms* or *arrows* (called *arr*(**A**)). Objects are usually denoted as

$$A, B, C, ..., X, Y, ...$$

and morphisms are usually denoted

$$f, g, h, ..., \alpha, \beta, ...$$

Further:

(1) Each morphism has a designated *domain* and *codomain* in *obj*(**A**). When the *domain* of f is A and the *codomain* of f is B we write $f: A \rightarrow B$.

(2) Given morphisms $f: A \rightarrow B$ and $g: B \rightarrow C$, there is a designated *composite* morphism

$$g \bullet f: A \rightarrow C.$$

* Department of Computer Science and Engineering, Božetěchova 2, CZ-612 66 Brno

(3) Given any object A there is a designated identity morphism 1_A: $A \to A$.

(4) The data above are required to satisfy the following:

Identity laws: If f: $A \to B$ then

$$1_B \bullet f = f \qquad \text{and} \qquad f \bullet 1_A = f.$$

Associative law: If f: $A \to B$, g: $B \to C$ and h: $C \to D$ then

$$h \bullet (g \bullet f) = (h \bullet g) \bullet f.$$

Although the category is described in terms of functions, it is important to realise that the notion of category is axiomatically defined and formal. The objects of a category need not be actual sets and the arrows need not be actual functions. As it is usual with axiomatic definitions this allows great flexibility in the construction of categories. This flexibility is crucial in the application of categories to modelling.

The principal axiom on a category is the associativity of composition. This axiom means that one can leave out brackets in a composition of arrows. For example we can write $l \bullet g \bullet h \bullet i \bullet j \bullet l$ instead of $(l \bullet ((g \bullet h) \bullet i) \bullet (j \bullet l))$.

In category theory we usually operate with diagrams. Every arrow has its representation as a diagram. So, for example arrow f: $S \to T$ should be drawn like this:

$$S \xrightarrow{\ f\ } T.$$

2.2 Properties of Objects in a Category

1.2.2 Initial and Terminal Object

Definition: An object I in a category is called *initial* if and only if for every object A of a category there is a unique morphism from I to A: $I \to A$.

A dual definition to the definition of initial object is the definition of terminal object.

Definition: An object T in a category is called *terminal* if and only if for every object A of a category there is a unique morphism from $A \to T$.

Category theory concludes following interesting properties (can be found in [1]):

- initial and terminal objects are unique up to isomorphism
- there exist categories without initial and/or terminal object

2.2.2 Products and Sums

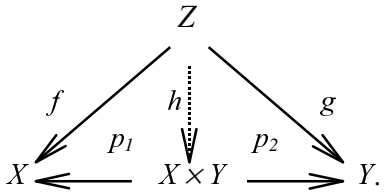
Definition: In a category A, an object $X \times Y$ with two morphisms (called projections)

$$X \xleftarrow{\ p_1\ } X \times Y \xrightarrow{\ p_2\ } Y$$

is called a *product* of X and Y if, given any object Z and morphisms f: $Z \to X$, g: $Z \to Y$, there is a unique morphism h: $Z \to X \times Y$ such that

$$p_1 \bullet h = f \qquad \text{and} \qquad p_2 \bullet h = g.$$

Diagrammatically,



Definition of sums is dual to the definition of products and is obtained when in the definition of a product all the morphisms are reversed. The formal definition of a sum follows.

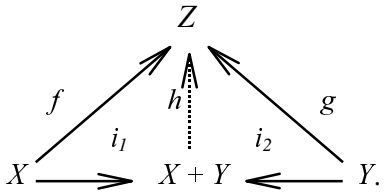
Definition: In a category A, an object $X + Y$ with two morphisms (called injections)

$$X \xrightarrow{i_1} X + Y \xleftarrow{i_2} Y$$

is called a *sum* of X and Y if, given any object Z and morphisms $f: X \to Z$, $g: Y \to Z$, there is a unique morphism $h: X \times Y \to Z$ such that

$$h \bullet i_1 = f \qquad \text{and} \qquad h \bullet i_2 = g.$$

In the form of a diagram:



Equivalently to the definition of initial and terminal objects, products and sums are unique up to isomorphism if they exist.

2.3 Mathematical Categories

Many objects in mathematics have been found to be categories. Typical and well-known category is a category of sets with (total) functions between them. This category is usually denoted as **Set**. As known, composition of functions between sets yields another function and for every set there is the identity function. Thus, the definition of being a category is satisfied.

In the category **Set**, the initial object is the empty set, the terminal object is every singleton. Products in **Set** are represented by a Cartesian products of pair of sets. Sums are expressed by a disjoint unions of sets.

Category **Set** is one of the simplest categories. A big class of categories is made up from sets with internal structure. Internal structure on a set, in mathematical sense, is given by additional conditions that have to be satisfied. A typical example of a set with structure is a group or monoid. The group is simply a set with binary operation on it, satisfying several conditions.

From universal algebra it is know the notion of a homomorphism. Homomorphism is simply a morphim preserving the structure of the set, i.e. for monoids, a homomorphism $f: (G_1,+) \to (G_2, *)$ has to satisfy $f(a + b) = f(a) * f(b)$.

Generally, structured sets and their homomorphisms form a category. Categories of structured sets are very usual in modelling and computer science.

2.4 The Usage of Categories in Modelling and Simulation

As in many other areas in computer science, even in modelling and simulation, there can be developed many categories. Usually, it is not necessary to introduce categories into theoretical framework of our work, but one can feel that categories are almost every time behind our ideas, theories and computations. Very often a chosen approach does not cross the limits of one category. In such cases introduction of categories into formal tool is not only useless but also ineligible.

On the other hand, the presence of categories allows one to switch the point of view on the solved problem or designed model. One might be able to set the model or theory into wide environment of other models and use the formalism of category theory to compare different approaches, models or to build up the link between different theories and models.

From this point of view, category theory is an opportunity, how to find a new expression of the problem solved in the context of other similar models. This link is developed by a strong formalism and it is possible to use it for further purposes, e.g. verification or analysis of models.

Categorical approach can be considered as an possible alternative way of describing the reality.

3 Models of Programming Languages

Programming language is a fundamental notion in modelling and simulation. When a new model is introduced, the simulation of such a model takes a form of a program written in a specially designed programming language.

Such a program is necessary to check, whether it satisfies all constraints necessary for a design of a correct model according to the specification. It means, that one needs to introduce a formal model of a programming language that will express all necessary features of a programming language, allows one to represent a model in quite natural way and still remains general and formal enough to allow an analysis.

The notion of a program equivalence is fundamental in the theory of programming languages. It is the semantic base for program optimisation, analysis of program fragments, verification, etc.

Program equivalence is typically defined according to following paradigm. Two program fragments are identified as being equivalent if and only if they have the same behaviour in all program contexts. Once a notion of a program equivalence has been selected for a programming language according to its semantics, its properties must be determined and analysis techniques found. The idea is to reduce the equivalence of program fragments (terms) to the equality in an appropriate model, i.e. to semantically capture the notion of program equivalence. It is then necessary to work with models that are correct in the sense that only equivalent terms are identified. Models with the ideal property that exactly the equivalent terms are identified are called fully abstract.

Several models of programming languages were introduced, some of them more general, some specially designed to suit purposes of a certain class of programming language. In next paragraphs the idea of a categorical model of a general programming language is presented. The possibility how correctness and full abstractness can be captured in the categorical background is also described. The construction is based on an idea of a category of many-sorted algebras and their homomorphisms.

Generally a programming language consists of a set of specified operators of specified types, which can be applied to the appropriate input values. Thus, the programming language is considered as a many sorted algebra over a signature that describes all the operators.

Definition: A *signature* Σ consists of a set of sorts S, a set of operators Σ_S and a function f: $\Sigma_S \rightarrow (S^* \times S)$, which assigns types to operators. Types $((s_1, \ldots, s_n), s')$ are usually written in the form $s_1 \times \ldots \times s_n \rightarrow s'$. Types $((), s')$ are usually written as constants of specified type s'.

Definition: Σ-algebra $\boldsymbol{A}$ is an S-indexed family of sets A (the carrier of $\boldsymbol{A}$) together with an operation $\sigma_{\boldsymbol{A}}$: $A_{s1} \times \ldots \times A_{sn} \rightarrow A_{s'}$, for each $\sigma \in \Sigma$ of type $s_1 \times \ldots \times s_n \rightarrow s'$.

A Σ-algebra is a natural definition of all possible program operators (terms). The idea of linking different program fragments is expressed by a homomorphism of Σ-algebras.

Definition: A homomorphism h: $\boldsymbol{A} \rightarrow \boldsymbol{B}$ over algebras is a function h: $A \rightarrow B$ such that for all $\sigma \in \Sigma$ of type $s_1 \times \ldots \times s_n \rightarrow s'$,

$$h_{s'}(\sigma_{\boldsymbol{A}}(a_1, \ldots, a_n)) = \sigma_{\boldsymbol{B}}(h_{s1}(a_1), \ldots, h_{sn}(a_n)), \text{ for all } a_i \in A_{si},\ 1 \leq i \leq n.$$

Now, we can construct a category of Σ-algebras:

Lemma: Σ-algebras together with their homomorphisms form a category.

Proof is elementary. Can be found in [4].

There exists one special algebra among the Σ-algebras called *term algebra.*

Definition: Term algebra $\boldsymbol{\tau_\Sigma}$ is a Σ-algebra such that its carrier T is least such that if $\sigma \in \Sigma$ has type $s_1 \times \ldots \times s_n \rightarrow s'$ and $t_i \in T_{si}$, $1 \leq i \leq n$, then $(\sigma, (t_1, \ldots, t_n)) \in T_{s'}$. If $\sigma \in \Sigma$ has type $s_1 \times \ldots \times s_n \rightarrow s'$ then the operation σ_τ is defined by $\sigma_\tau(t_1, \ldots, t_n) = (\sigma, (t_1, \ldots, t_n))$.

The term algebra represents all syntactic structures of a selected programming language. It is fully syntactic structure without any notion of semantics. For example, assume an operator $+$: $s_i \times s_i \rightarrow s_i$ and $\{1, 3\} \subset T_{si}$, then $(+, (1, 3)) \in T_i$, $(+, (1, (+, (1, 3)))) \in T_i$, etc.

Lemma: The term algebra $\boldsymbol{\tau_\Sigma}$ is the initial object in the category of algebras and their homomorphisms (proof can be found in [4]).

As known from category theory there exist an unique morphism from the term algebra to any other Σ-algebra. This homomorphism from $\boldsymbol{\tau_\Sigma}$ to $\boldsymbol{A}$ is denoted $M_{\boldsymbol{A}}$.

$M_{\boldsymbol{A}}$ is a special homomorphism. It goes from fully syntactic algebra. It can be thought as the meaning or semantic function from syntax to semantics. We reached the representation of a mapping between syntax and semantics in a natural way in a category. Now, we focus on representation of correctness and abstractness of models.

Recall that every function $f: D \to E$ over sets D, E induces an equivalence relation over D, denoted as $\equiv_f$ that is defined by: $d_1 \equiv_f d_2 \Leftrightarrow f(d_1) = f(d_2)$. If f is an homomorphism then $\equiv_f$ is a congruence relation.

Definition: For an algebra $\boldsymbol{A}$, the congruence over τ_Σ induced by $M_{\boldsymbol{A}}$ is denoted as $\approx_{\boldsymbol{A}}$.

The congruence $\approx_{\boldsymbol{A}}$ represents the equal program fragments. These fragments might be substituted for each other without the change of program behaviour. This equivalence is defined on the syntactic structure of program terms.

According to the program semantics (denotational or operational), it is possible to introduce another congruence relation over τ_Σ. We will denote this congruence relation as $\approx$.

Correctness and full abstractness is expressed in the presented categorical model of a programming language as follows.

Definition: Model $\boldsymbol{A}$ is called $\approx$*-correct* if and only if $\approx_{\boldsymbol{A}} \subseteq \approx$. Model $\boldsymbol{A}$ is called $\approx$*-fully abstract* if and only if $\approx_{\boldsymbol{A}} = \approx$.

Other results based on the presented theory are discussed in [4].

4 Categorical Models of Behavioural Equivalence of State-Based Models

Another nice area of modelling which can be represented categorically, is the behavioural equivalence of state-oriented models. Several models of concurrent and sequential behaviour have been introduced, e.g. Petri nets, transition systems, process algebras.

The crucial notion in the theory of behaviour is the equivalence of state-based models. The idea of behavioural equivalence was approached by several authors in several ways. The most complex theory was presented by Milner (see [2]).

Today, the idea of heterogeneous models is on the issue. The lack of analytical and verification techniques creates a sequence of serious problems. In huge systems, the communication of several independent parts is hard to observe without any general methodology. The necessity of mathematical background is more than evident. Mathematical foundations can allow one to decide about the right functionality of models before they are designed into a bigger system.

Also this thought might be accessed from categorical point of view. The presented approach is set up by the theory of bisimulation of systems, especially on a strong bisimulation.

The concept of strong bisimulation appeared in Milner's work (see [2]) in the context of CCS calculus. However, this idea is more general and can be transferred into Petri net and automata based models. We will present the categorical idea of behavioural equivalence for A-labelled P/T Petri nets (see [5]).

Consider two Petri nets PN_1 and PN_2 in which transitions are labelled via functions λ_1 in PN_1 and λ_2 in PN_2 by the elements of a fixed alphabet **A**. Such nets are called A-labelled. Assuming that P_1, P_2 are set of places of the PN_1, PN_2; T_1 and T_2 are set of transitions and M_1, M_2 are set of reachable markings (the definition does not depend on initial marking).

Then, the bisimulation relation μ*:* $PN_1 \to PN_2$ is a binary relation μ, $\mu \subseteq M_1 \times M_2$ with the following property (bisimulation law):

If $(m_1, m_2) \in \mu$ ($m_1 \in M_1$ and $m_2 \in M_2$) then:

(1) whenever for some $t_1 \in T_1$ holds $m_1[t_1 > m_1'$ $(m_1' \in M_1)$ then there exist $t_2 \in T_2$, $\lambda_1(t_1) = \lambda_2(t_2)$ such that $m_2[t_2 > m_2'$ $(m_2' \in M_2)$ and $(m_1', m_2') \in \mu$.

(2) whenever for some $t_2 \in T_2$ holds $m_2[t_2 > m_2'$ $(m_2' \in M_2)$ then there exist $t_1 \in T_1$, $\lambda_1(t_1) = \lambda_2(t_2)$ such that $m_1[t_1 > m_1'$ $(m_1' \in M_1)$ and $(m_1', m_2') \in \mu$.

For a definition of a category it is better to start with a weaker condition and to develop the bisimulation relation further. This weaker condition is the condition of being simulation relation. We say that a binary relation is a simulation relation, when it satisfies everything from the definition of bisimulation relation except the condition (2).

When $\mu: PN_1 \rightarrow PN_2$ is a bisimulation relation, we say that system PN_2 simulates system PN_1. One can feel that bisimulation relations are special type of simulation relations. This gets the conclusion, which is easy to prove:

μ is a *bisimulation relation* $\Leftrightarrow$ μ and μ^T are both simulation relations, where $\mu^T = \{(x, y).\ (y, x) \in \mu\}$, $\mu^T: PN_2 \rightarrow PN_1$.

Now, it is possible to define the simulation category of A-labelled Petri nets as follows:

objects are A-labelled Petri nets,
morphisms are simulation relations,
composition of morphisms is defined as composition of relations,
identity is the diagonal relation $1_M = \{(x, x).\ x \in M\}$.

It is necessary to check properties of being a category. This yields immediately from the properties of simulation relations (see [2] or [5]).

When bisimulation is on the issue, it is easy to show that bisimulation category of A-labelled Petri nets is a subcategory of the simulation category of A-labelled Petri nets when morphisms are such simulations whose transposed morphisms (relations) are simulation morphism either, going in the other way.

Category theory seems to be usable in transferring notions of behavioural equivalence for Petri nets and similarly, the presented approach might be used for other models of concurrency as transition systems or CCS.

The category of A-labelled Petri nets is a category of P/T Petri nets equipped by a labelling of transitions. In the area of modelling, many other extensions of Petri nets classes are used, e.g. coloured Petri nets or object-oriented Petri nets. Several attempt can be found in the literature to develop categorical theory even for such Petri nets classes.

Categories provide nice tool for models with relevant algebraic representation. In many applications, the concept of Petri nets as bipartite graphs equipped by transition labelling, timing, etc., is not suitable. From this reason other algebraic representations of certain Petri net classes are discussed by other researchers.

When object-oriented approach is used, the structure of the object-oriented Petri nets is even more sophisticated. Incorporating the programming language theory, as discussed above, seems to be promising for this research to the future.

5 Conclusions

The mathematical concept of a category is a flexible idea. With this in mind, one can incorporate categories into the done work and build up a different point of view of a specified topic. Category theory offers its well-known background to other areas. The main aim of this paper was to show, that the categorical concept is also natural for modelling and simulation. However, categories have their advantages and disadvantages as almost everything, but still can provide an general formal tool for researchers.

This work has been supported by the Grant Agency of Czech Republic under the contract No. 102/98/0552 "Research and Applications of Heterogenous Models" and by the Research intention No. CEZ: J22/98: 262200012 – „Research in information and control systems“.

References

1. Lawvere F. W., Schanuel S. H.: *Conceptual Mathematics: A first introduction to categories*, Cambridge University Press, 1997 (translated from Italian edition 1991).
2. Milner R.: *Communication and Concurrency*, Prentice Hall (UK), 1989.
3. Peterson J. L.: *Petri Net and the Modelling of Systems*, Prentice Hall (USA), 1981.
4. Stoughton A.: *Fully Abstract Models of Programming Languages*, Pitman, London, 1988.
5. Urbášek M.: *Bisimulation relations in categories of A-labelled Petri nets and the connection of CCS and Petri nets*, Technical report, Department of Comp. Science and Engineering, FEECS, Brno University of Technology.