Formalization of Agents and Multi-Agent Systems. The Special Case of Category Theory Working Paper*

Giovanna Di Marzo Serugendo Murhimanya Muhugusa
Christian Tschudin†
Jürgen Harms

Centre Universitaire d'Informatique, University of Geneva
24, rue Général Dufour, CH-1211 Genève 4
Phone: +41 22 705 76 43, Fax: +41 22 705 77 80
e-mail: dimarzocui.unige.ch
url: http://www.unige.ch/ dimarzo/home.html

Cahier du CUI no 109

December 16, 1996

*This work is supported by the Swiss National Fund for Scientific Research (FNSRS) grant 20-40631.94

†Institute fur Informatik, University of Zurich

Abstract

This working paper lists some ideas on how to apply category theory to agents, and multi-agent systems. It focuses more on the distributivity and compositionality of agents, as well as on the emergence of sociality and properties, than on the intelligence or on the ability of agents to reason.

It presents some categorical notions, and explains informally how they can be connected with agents and multi-agent systems.

Contents

1 Introduction

Specifications and formal representations of agents and/or multi-agent systems (MAS) are represented by the means of temporal logics and speech acts inside the Distributed Artificial Intelligence community. Temporal logics and speech act help for representing the reasoning of the agents and their communications respectively. The Petri Net community is more concerned with the distributivity of agent-based applications.

Category theory is now widely used inside the petri net community for expressing the semantics of the whole staff of petri nets types, for comparing them on the basis of this semantics, for composing (by fusion or refinement) two or more petri nets, etc. Little work is undertaken, in order to use category theory in the agent domain. Of course, category theory, being a very abstract tool, will not help formalizing specific algorithms, or the reasoning of agents. Obviously, it is not possible to capture all features of all MAS. We are more interested in capturing the cooperative and compositional aspect of MAS in the framework of category theory, than the intelligence that agents may have.

This working paper lists some ideas on how to apply category theory to isolated entities (agents), or to multiple entities (distributed systems and MAS). We are particularly interested in the following points:

- *Cooperation/Composition.*
  What is the mathematical object corresponding to the cooperation or composition of multiple software agents? What is the categorical definition of cooperation, collaboration and communication between components?

- *Properties of Structured Specifications.*
  What about the properties of a structured (horizontally or vertically composed) specification, how is it possible to change a component (an agent) without changing the specification properties?

- *Primitives of Agents and* MAS.
  How to give a categorization of (some of) the primitives of agents and groups of agents?

- *Emergence of properties and Emergence of Sociality [26, 12]*
  A property has *emerged* when it is there, in the MAS, and we have not specified it. Social emergence can be seen as the groups of agents formed by communicating agents, by collaborating agents. As for emergence of property, social emergence is *observed* but not necessarily requested by the specifier.
  What is the categorical definition of properties emerging in a multi-agent system and of families of agents appearing in a multi-agent system? The characteristics of such a system are distributivity (in space and time), concurrency, cooperation, communication, dynamicity (new agents appear, old agents disappear), dynamic changing set of agents composing a family (agents enter and retreat into/from families).

- *Global state of a distributed system*
  How is it possible to determine the global state of a distributed system, when we know only local states [2, 21]?

This paper firstly presents primitives for both agents and multi-agents systems together with a brief list of theories of agents and MAS. It then mentions some uses of category theory in diverse fields of computer science. Finally it clarifies what we intend by agents and families of agents, and proposes how some categorical elements can be used for defining or capturing properties of multi-agent systems, i.e., we will explore several categorical objects and see how they can be applied to agents, multi-agent systems, distributed systems in general.

2 Agents and Multi-Agent Systems

2.1 Definitions and Primitives of Agents

What is an agent is a controversial question, and is still an open question inside the agent community. Different philosophies arose: (1) an intelligent entity (capable of reasoning) is an agent; (2) any program is an agent, because it realizes some functionality on a user's behalf; (3) between these two extremes a graduation in the agenthood has been proposed in order to evaluate each program in an agent fashion; (4) depending on its environment an agent may have or not some capabilities and primitives, e.g. depending on the environment an agent has to be mobile.

The controversial question is around the set of primitives that an agent must have to be authorized to be called an agent. We will not try to give our definition, we will only give a list of the possible agents' primitives [28, 14].

- *Autonomy*
  Agents operate without the direct intervention of humans and have some kind of control over their actions and internal state;
- *Social Ability*
  Agents communicate with each other;
- *Reactivity*
  Agents reacts on the impulse they receive from their environment;
- *Pro-activeness*
  Agents are able to take initiative;
- *Mentalistic/Intentional notions*
  There are two groups of intentional properties: (1) the information attitudes: belief, knowledge and (2) the pro-attitudes: desires, intentions, obligations, choice, etc. Information attitudes represent the information an agent has about its environment, while the pro-attitudes capture what an agent wants to do in order to reach its goal;
- *Emotion*
  Emotion is a characteristic which lets us believe more in an agent, especially for user interfaces, it touches anthropomorphism [4, 5];
- *Mobility*
  Agents are able to travel across a network;
- *Veracity*
  Agents communicate true information;

- *Benevolence*
  Agents do not have conflicting goals and always try to do what they are asked;

- *Rationality*
  Agents will act to achieve their goal and will not act in order to prevent their goal to be achieved;

- *Successful*
  Agents accomplish their task;

- *Capable*
  Agents possess the effectors needed to accomplish the task; e.g. the softbot [11] uses FTP or telnet as effectors to retrieve/handle documents;

- *Perceptive*
  Agents use sensors in order to distinguish characteristics of the world allowing them to use their effectors to achieve the task: a robot has sensors for hearing of viewing obstacles, while softbots sensors are net finders;

- *Deliberative*
  Agents maintain an internal model of the world and reason about the results of their actions. They choose the actions according to their world model.

- *Predictive*
  The world model allows the agents to predict correctly the actions to perform in order to achieve the task.

- *Interpretive*
  Agents correctly interpret its sensor.

- *Rational*
  Agents choose the actions that it predicts.

- *Sound*
  Agents are predictive, interpretive and rational.

Agents are of different types, depending on which primitives they are able to exhibit or realize. We can say that an agent has a goal and takes some *actions* in order to reach its goal. Actions are decided according to the type of the agent.

2.2 Definitions and Primitives of Multi-agent systems

Primitives of multi-agent systems are of two types: (1) the set of agents' primitives extended to groups of agents, and (2) interaction primitives between agents.

- *Agents' primitives* extended to MAS.

- *Cooperation, collaboration, negotiation*
  Agents may have common or separate goals, in order to realize them, they need to interact with each other.

- *Communication*
  In a MAS, agents communicate, usually by message passing in order to cooperate or negotiate. In Speech Act communication is considered as an action.

- *Coordination*
  Coordination between agents is used to ensure that the actions of the agents in a system do not conflict and are not self-defeating.

- *Coherence*
  The global system performance is satisfactory.

Agents involved in a MAS maintain a partial knowledge of the whole system, communication is then necessary to provide a means of exchanging information among agents in order to maintain the coordination and coherence of the whole MAS.

2.3 Agent Theories

The formalization of agents is concerned with (1) the specification of the agents' primitives and (2) the specification of the agents' reasoning about primitives.

From the properties listed in 2.1, it appears [28] that the intentional ones help describe almost everything, even "inert" objects like light switch. For this reason, a lot of theories have arisen around the capture of and the reasoning about the intentional properties and how these properties induce an agent to take an action. As the AI community is primarily concerned with agents, particularly with intelligent ones, great importance is given to *reasoning*. The reasoning of an agent about what it knows and what actions it has to take, the reasoning of an agent about the other agents, etc. Thus, much theories are concerned with the formalization of agent properties and agent reasoning.

Besides intentional logic, some alternative approaches are use to formalize agents. Below we give a brief list of some of classical and alternatives approaches:

- *Intentional Logic*
  As agents are primarily studied in the AI field, classical approaches are symbolic/logic ones. They are based on the representation of the agents' intentions with formulae of some logical language [25, 29].

- *Game Theory*
  Game theory is used as a strategy to help agents to take decisions.

- *Subsumption Architecture*
  Some other radical approach, opposed to AI is given by the subsumption architecture of Brooks. Brooks, in [6, 7] argues that intelligence is an *emergent* property of systems, and that there is no need of symbolic AI for an explicit representation and an explicit abstract reasoning of intelligent behavior.

- *Tasks and environment*
  Other approaches consider the behavior of the agent, instead of its intentions. Success is determined upon the observable behavior of the agent. Thus, observing the behavior avoids observing the intentional attitudes.

These approaches are based on the formalization of the agents' properties related to tasks and environment.

ATAL proceedings [31, 30] covers the current research on this topic.

Specifications of some agents' primitives are formalized by Goodwin in [14], he uses the Z specification language in order to clarify the definition of the properties related to the behavioral aspect of agents.

2.4 Multi-Agent Systems Theories

The formalization of multi-agent systems is concerned with (1) the specification of agents' properties, (2) the specification of agents' communications and (3) the reasoning about global consistency of the whole system.

Formal theories for social agencies intend to furnish methods for reasoning about communication among agents and methods for the building of social agents. Speech act is the most widely used method for reasoning about communication among agents.

Formalization of social agencies are concerned with the formalization of agents' intentions and communications between agents. Formalisms of MAS are extended from formalisms for agents. They capture both the intentional notions and the communications among agents. Classically a unified logical formalism is used for capturing both the mentalistic notions of the agents, and the communications among them. Currently the communication part is based on speech act. Some work [29, 25] combine in a logical framework the temporal logic and speech act part.

MAAMAW proceedings [26] covers the current research on multi-agent systems.

Besides intentional logics and speech act, some other alternatives arise.

- *Temporal Logic and Speech Act*
  There are two types of temporal logic: branching time temporal logic and linear time temporal logic. Communication between agents inside a multi-agent system is based on speech act performatives. Performatives are considered as actions and the set of actions is extended to incorporate the performatives [25, 29].

  Other temporal logics that are emerging are: Stubborn sets, Interleaving Set Temporal logic [23].

- *Linear Logic*
  Cap in [10] provides a calculus for distributed parallel processes which is a mixing of linear logic and algebraic specifications.

- *Formal Methods for Describing Concurrency*
  We can mention classical formal methods adapted to distributivity and concurrency: Finite State Automata, Petri Nets, Process Algebra (CSP, CCS, LOTOS), Event Structures, Graph Grammars.

- *Petri Nets applied to Agents*
  More and more work is realized in applying Petri Nets to agents, multi-agents, and cooperative systems [24, 1].

- *Category Theory*
  Cap [9] has demonstrated that a parallel non-deterministic distributed process can be given an operational semantics in terms of a Symmetric Monoidal Category (SMC).
  A transition is a modification of the state of the information (not of the state of the agents). Objects in the Category are states of the information and morphisms are transitions between these states. Parallelism is given by a bifunctor which combines two morphisms and their associated domains and codomains. Sequentiality is given by the composition of morphisms.

3 Category Theory

To intuitively understand category theory, the notion of structured set is very helpful to keep in mind. Consider a set provided with a *structure*, e.g. a group, a ring, a monoid, etc. Different sets can have different structures but of the same kind. (i.e. two groups with their own group structure); or two different structures of the same kind can be applied to a same set (i.e. a set with two different orders). A structured set is given by the set and the structure applied on the set (E, S). For a given *kind of structure*, and for each pair of structured sets $(E_1, S_1), (E_2, S_2)$, it is possible to define a set of applications from (E_1, S_1) to (E_2, S_2), which are compatible with the structure (i.e. group homomorphisms preserve group structure). Compatible means that for all applications f from (E_1, S_1) to (E_2, S_2) and for all applications g from (E_2, S_2) to a third structured set (E_3, S_3), $g \circ f$ is an application from (E_1, S_1) to (E_3, S_3).

The classes, whose objects are all sets of a given structure and whose morphisms are all applications preserving the structure, are categories (i.e. the category of all groups as objects and group homomorphisms as morphisms).

A category has a structure given by its arrows, functors preserve this structure.

The set of all sets does not exists, but the category of all categories do exist. Categories are used when sets are not sufficient.

Category theory is a more abstract language for defining objects. It is essentially used for describing objects (in computer science, in nature, etc) as mathematical entities. It gives an abstraction to considered concepts, and enables us to reason about them. Two concepts which seems not to be related in a concrete world, appear to have common abstract properties when they are considered in a categorical point of view. Category theory provides a method for abstract specifications: (1) specify abstractly the program and the data types before the implementation, (2) properties are defined element-free.

[3] and [16] provide the necessary background for category theory.

This section firstly presents already well known applications of category theory in computer science, and then gives some formal categorical definitions that will be informally explained and applied to agents in section 5.

3.1 Category Theory in Computer Science

Category theory is an abstract mathematical tool which is now widely used in several computer science domains.

- *Petri Nets*
  Category theory is now commonly used inside the Petri net community, essentially to give a semantics to Petri Nets and to all possible extensions of Petri Nets, for composing (by fusion or refinement) two or more petri nets [8, 17, 20, 15].

- *Programming Languages, Logic*
  Programming language specification and semantics, lambda-calculus, etc. are more and more investigated under a categorical point of view. For transition systems: the observation starts from considering that the states of the transition system are the objects of the category, and the transitions between two states are the morphisms of the category. For logic, the formulae of the logic are the objects and the proof of formulae are the morphisms. CTCS [22] lecture notes covers the current research on category theory applied to computer science.

- *Category Theory in distributed systems/multi-agent systems*
  Cap [9] has demonstrated that a parallel non-deterministic distributed process can be given an operational semantics in terms of a Symmetric Monoidal Category (SMC).

3.2 Definitions

This section provides the categorical definitions we use in section 5.

Category

Definition 3.1 *Function [3].*
A function f *is a mathematical entity with the following properties:*

1. f *has a domain* S *and a codomain* T*, both are sets, it is denoted by* $f : S \to T$
2. $\forall x \in S, f$ *has a value at* x*, which is an element of the codomain and is denoted* $f(x)$
3. *The domain, the codomain, and the value* $f(x)$ *for each* x *in the domain are all determined completely by the function, and conversely they completely determine the function* f

Two functions f, f' with the same domain S, but with two different codomains T, T' are two different functions, even if $f(x) = f'(x)$ for each $x \in S$.

Definition 3.2 *Mapping [3].*
A mapping *is a relation* $R \subseteq S \times T$ *with the functional property:*

$$\textit{for each } s \in S, \textit{ there exists a unique } t \in T \textit{ s.t. } (s,t) \in R$$

A mapping completely defines the domain S but not the codomain T, so that contrarily to the functions, two mappings with different codomains can be equal. Moreover, in the definition of mappings S, T do not need to be sets.

Definition 3.3 *Graph of a function [3].*
Let $f : S \to T$ *be a function, the* graph of f *is the set of ordered pairs* $\{(x, f(x)) \mid x \in S\}$.

The graph of a function is a relation from the domain to the codomain of the function with the functional property.

Definition 3.4 *Graph [3].*
A graph $\mathcal{G}$ *is given by:*

1. G_0, *a collection of nodes (or objects),*
2. G_1, *a collection of arrows (or morphisms),*
3. *Two functions* dom, cod
$$G_1 \underset{cod}{\overset{dom}{\rightrightarrows}} G_0$$

Each arrow f must have a source (domain) node a and target (codomain) node b, it is denoted by $f : a \rightarrow b$ where $dom(f) = a, cod(f) = b$.

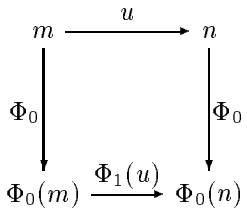
The so defined graphs are multidirected graphs with loops: there can be zero, one or more arrows having the same source and/or target nodes.

Remark 3.5 *The collection of nodes and arrows of a given node do not necessarily form sets. If it is the case the graph is called* small graph, *otherwise it is called* large graph.

Definition 3.6 *Graph Homomorphism [3].*
Let $\mathcal{G}, \mathcal{H}$ *be two graphs, a graph homomorphism* $\Phi : \mathcal{G} \rightarrow \mathcal{H}$, *is a pair of mappings:*

1. $\Phi_0 : G_0 \rightarrow H_0$
2. $\Phi_1 : G_1 \rightarrow H_1$
3. *With the property that for each arrow* $u : m \rightarrow n \in G$, *then* $\Phi_1(u) : \Phi_0(m) \rightarrow \Phi_0(n)$

A graph homomorphism preserves the arrows between the nodes. We can also depict the third condition with the following commutative diagram:



Definition 3.7 *Composable Pairs [16].*
Let $\mathcal{G}$ *be a graph, with* G_0 *the set of nodes,* G_1 *the set of arrows. The set of composable pairs of arrows is the set:*

$$G_2 = \{(g, f) \mid g, f \in G_1 \wedge dom(g) = cod(f)\}$$

A pair of arrows (g, f) is composable if the codomain of f is the domain of g.

Notation 3.8 *Given a graph* $\mathcal{G}$, *we denote by* G_0 *the collection of nodes (or objects) of the graph, by* G_1 *the collection of arrows (or morphisms) of the graph, and by* G_2 *the collection of composable pairs of arrows of* G. *This convention applies also for categories.*

A category is a graph with the possibility to compose arrows having corresponding nodes.

Definition 3.9 *Category [16].*
A category $\mathcal{C}$ is a graph with two additional functions:

1. $C_0 \xrightarrow{id} C_1$ *the identity*
   $c \mapsto id_c$
2. $C_2 \xrightarrow{\circ} C_1$ *the composition*
   $(g, f) \mapsto g \circ f$.

Verifying the following properties:

1. $\forall a \in C_0 \quad \Rightarrow dom(id_a) = cod(id_a) = a$
2. $\forall (g, f) \in C_2 \quad \Rightarrow dom(g \circ f) = dom(f), cod(g \circ f) = cod(g)$
3. $(g, f), (k, g) \in C_2 \quad \Rightarrow k \circ (g \circ f) = (k \circ g) \circ f$, *associativity*
4. $\forall f : a \to b \in C_1 \quad \Rightarrow f \circ id_a = id_b \circ f = f$, *unit law*

A category is a graph with identity arrows and a composable associative operation $\circ$ between arrows such that identity arrows are neutral elements for $\circ$.

The category of sets **Set** has sets as objects and functions between sets as arrows.

Definition 3.10 *Hom [3].*
Let $\mathcal{C}$ be a category, $a, b \in C_0$ two objects of the category, the set of all morphisms f from a to b is given by:

$$Hom_{\mathcal{C}}(a, b) = \{f \in C_1 \mid dom(f) = a, cod(f) = b\}$$

Definition 3.11 *Terminal and Initial Objects [3].*
An object of a category $\mathcal{C}$ is called terminal *and is noted* 1 *if:*

$$\forall A \in C_0 \Rightarrow \exists! f : A \to 1 \qquad \begin{matrix} A_1 \longrightarrow \\ A_2 \longrightarrow \\ \cdots \end{matrix} 1$$

An object of a category $\mathcal{C}$ is called initial *and is noted* 0 *if:*

$$\forall A \in C_0 \Rightarrow \exists! f : 0 \to A \qquad 0 \begin{matrix} \longrightarrow A_1 \\ \longrightarrow A_2 \\ \cdots \end{matrix}$$

Definition 3.12 *Monics and Epis [3].*
Let $\mathcal{C}$ be a category, an arrow $f : A \to B \in C_1$ is a monomorphism (or monic) *if:*

$$\forall T \in C_0, \forall f_1, f_2 : T \to A \Rightarrow (f \circ f_1 = f \circ f_2 \Rightarrow f_1 = f_2)$$

$$T \underset{f_2}{\overset{f_1}{\rightrightarrows}} A \xrightarrow{f} B$$

Let $\mathcal{C}$ be a category, an arrow $f : A \to B \in C_1$ is an epimorphism (or epi) *if:*

$$\forall T \in C_0, \forall f_1, f_2 : B \to T \Rightarrow (f_1 \circ f = f_2 \circ f \Rightarrow f_1 = f_2)$$

$$A \xrightarrow{f} B \underset{f_2}{\overset{f_1}{\rightrightarrows}} T$$

Functors

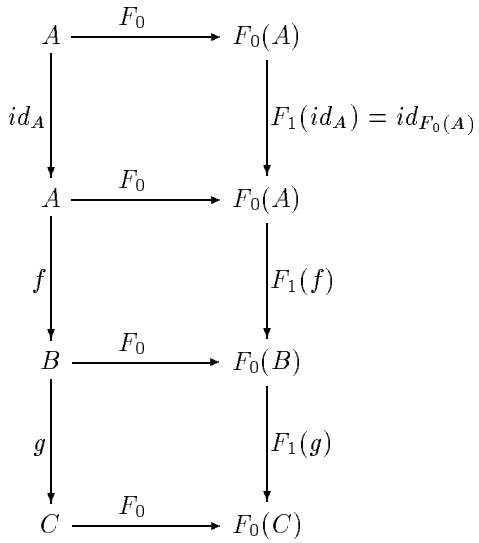
Definition 3.13 *Functor [3].*
Let $\mathcal{C}, \mathcal{D}$ be two categories, a functor $F : \mathcal{C} \to \mathcal{D}$ *is a pair of functions:*

1. $F_0 : C_0 \to D_0$
2. $F_1 : C_1 \to D_1$

such that:

1. $\forall f : A \to B \in C_1 \quad \Rightarrow F_1(f) : F_0(A) \to F_0(B) \in D_1$
2. $\forall A \in C_0 \quad \Rightarrow F_1(id_A) = id_{F_0(A)}$
3. $\forall g \circ f \in C_1 \quad \Rightarrow F_1(g \circ f) = F_1(g) \circ F_1(f) \in D_1$

That is to say that the following diagram commutes:



Bifunctor

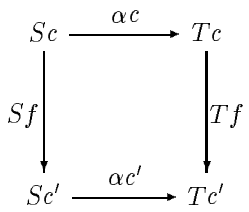
Definition 3.14 *Bifunctor [16].*
A functor $F : \mathcal{B} \times \mathcal{C} \to \mathcal{D}$ from a product category is called a bifunctor.

The objects of a product category $\mathcal{B} \times \mathcal{C}$ are pairs of objects $< b, c >$ of objects b of $\mathcal{B}$ and c of $\mathcal{C}$. The arrows of a product category $\mathcal{B} \times \mathcal{C}$ are pairs of arrows in each category $\mathcal{B}$ and $\mathcal{C}$.

Natural Transformations

Definition 3.15 *Natural Transformation between Functors [16].*
Given two functors $S, T : \mathcal{C} \to \mathcal{B}$, a natural transformation $\alpha : S \to T$ is a function which assigns to each object c of $\mathcal{C}$ an arrow $\alpha c : Sc \to Tc$ such that

every arrow $f : c \to c'$ in $\mathcal{C}$ the diagram



commutes.

The composite $\beta \circ \alpha$ of two natural transformations β and α is defined componentwise: $(\beta \circ \alpha)a = \beta a \circ \alpha a$. The composite of two natural transformations is also a natural transformation. It yields to the category of functors, where objects are functors and arrows are natural transformations between functors.

Commutative Diagrams

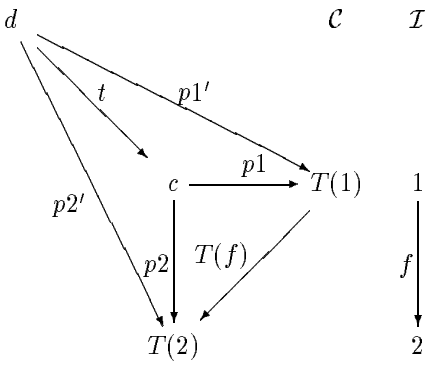
Definition 3.16 *Diagram [3].*
Let $\mathcal{G}$, $\mathcal{H}$ be two graphs, a diagram in $\mathcal{H}$ of shape $\mathcal{G}$ is a homomorphism: $D : \mathcal{G} \to \mathcal{H}$.

Definition 3.17 *Commutative Diagram [3].*
Let $D : \mathcal{G} \to \mathcal{H}$ be a diagram, and $\mathcal{H}$ be the underlying graph of a category, also called $\mathcal{H}$. D is commutative if for any nodes i, j of $\mathcal{G}$ and two paths $s_1 \ldots s_n$, $l_1 \ldots l_m$ from i to j in $\mathcal{G}$, the two paths $D(s_1) \circ \ldots \circ D(s_n)$ and $D(l_1) \circ \ldots \circ D(l_m)$ compose to the same arrow in category $\mathcal{H}$.

Products and Sums

Definition 3.18 *Limit.*
Let $\mathcal{C}$ be a category, let $\mathcal{I}$ be a small-category and $T : \mathcal{I} \to \mathcal{C}$ be a covariant functor. A **limit** *of T consists of an object $c \in Ob(\mathcal{C})$ together with a family of morphisms $\{pi : c \to T(i)\}_{i \in \mathcal{I}}$ s.t.*

- $\forall f : i \to j, f \in Mor(\mathcal{I})$, $T(f) \circ pi = pj$ *(1)*
- $\forall d \in Ob(\mathcal{C})$ *together with* $\{pi' : d \to T(i)\}_{i \in \mathcal{I}}$ *satisfying (1),* $!\exists t : d \to c$ *s.t.* $pi \circ t = pi'$, $\forall i \in \mathcal{I}$.



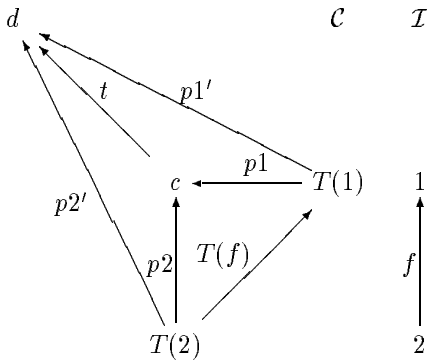
In ***Set****: Biggest set*

Definition 3.19 *Colimits.*
Let $\mathcal{C}$ be a category, let $\mathcal{I}$ be a small-category and $T : \mathcal{I} \to \mathcal{C}$ be a covariant functor. A **colimit** *of T consists of an object $c \in Ob(\mathcal{C})$ together with a family*

of morphisms $\{pi : T(i) \to c\}_{i \in \mathcal{I}}$ *s.t.*

- $\forall f : j \to i, f \in \mathcal{I}$, $pi \circ T(f) = pj$ *(1)*
- $\forall d \in Ob(\mathcal{C})$ *together with* $\{pi' : T(i) \to d\}_{i \in \mathcal{I}}$ *satisfying (1),* $!\exists t : c \to d$ *s.t.* $t \circ pi = pi'$, $\forall i \in \mathcal{I}$.

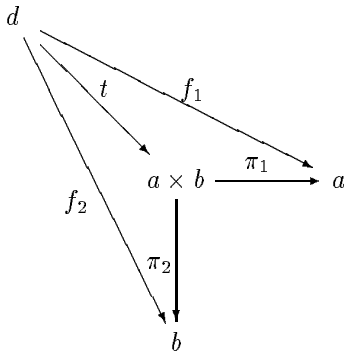
In ***Set****: Least set*



Definition 3.20 *Product.*
Given $\mathcal{C}$ *a category, let* $a, b \in Ob(\mathcal{C})$ *then the* **product** *of* a, b *is an object* $a \times b \in Ob(\mathcal{C})$ *together with 2 arrows* π_1, π_2 *s.t.*

- $\pi_1 : a \times b \to a, \pi_2 : a \times b \to b$
- $\forall d \in Ob(\mathcal{C})$ *with* $f_1 : d \to a$, $f_2 : d \to b$, $!\exists t : d \to a \times b$ *s.t.* $\pi_i \circ t = f_i$

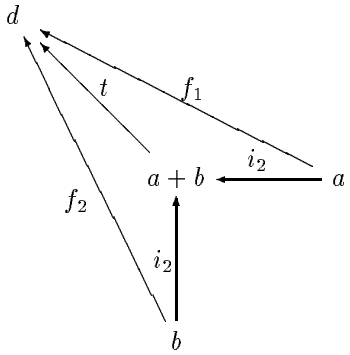
In ***Set****: Cartesian product*



Definition 3.21 *Coproduct.*
Given $\mathcal{C}$ *a category, let* $a, b \in Ob(\mathcal{C})$ *then the* **coproduct** *of* a, b *is an object* $a + b \in Ob(\mathcal{C})$ *together with 2 arrows* i_1, i_2 *s.t.*

- $i_1 : a \to a + b, i_2 : b \to a + b$
- $\forall d \in Ob(\mathcal{C})$ *together with* $f_1 : a \to d, f_2 : b \to d$ $!\exists t : a + b \to d$ *s.t.* $t \circ i_i = f_i$

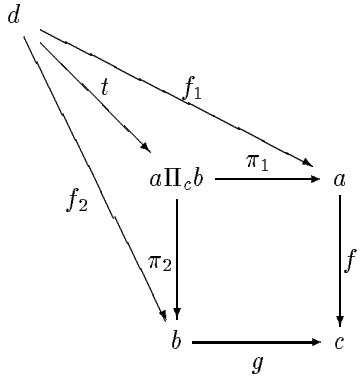
In ***Set****: Disjoint Union*



Definition 3.22 *Pullback.*
Given $\mathcal{C}$ *a category, let* $a, b, c \in Ob(\mathcal{C})$ *and* $f : a \to c$, $g : b \to c$, *then the*

pullback *of a, b is an object $a\Pi_c b \in Ob(\mathcal{C})$ together with 2 arrows π_1, π_2 s.t.*

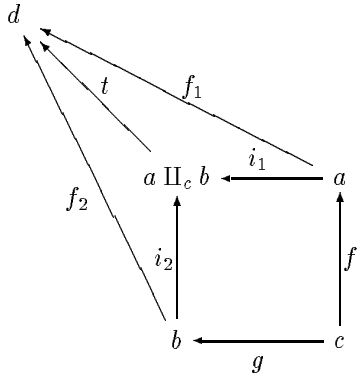
- $\pi_1 : a\Pi_c b \rightarrow a, \pi_2 : a\Pi_c b \rightarrow b$
  $f \circ \pi_1 = g \circ \pi_2$
- $\forall d \in Ob(\mathcal{C})$ *with* $f_1 : d \rightarrow a$, $f_2 : d \rightarrow b$, *s.t.*
  $f \circ f_1 = g \circ f_2$,
  $!\exists t : d \rightarrow a\Pi_c b$ *s.t.* $\pi_i \circ t = f_i$



In ***Set****: Pullback gives decomposition into objects having same image under f, g*

Definition 3.23 *Pushout.*
Given $\mathcal{C}$ a category, let $a, b, c \in Ob(\mathcal{C})$ and $f : c \rightarrow a$, $g : c \rightarrow a$, then the **pushout** *of a, b is an object $a \amalg_c b \in Ob(\mathcal{C})$ together with 2 arrows i_1, i_2 s.t.*

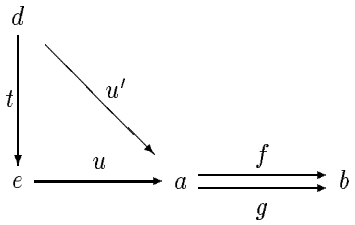
- $i_1 : a \rightarrow a \amalg_c b$, $i_2 : b \rightarrow a \amalg_c b$
- $\forall d \in Ob(\mathcal{C})$ *together with* $f_1 : a \rightarrow d, f_2 : b \rightarrow d$
  $!\exists t : a \amalg_c b \rightarrow d$ *s.t.* $t \circ i_i = f_i$



In ***Set****: Pushout gives composition of objects having same preimage under f, g*

Definition 3.24 *Equalizer.*
Given calC a category, let $a, b \in Ob(calC)$ and $f, g : a \rightarrow b$, then the **equalizer** *of a, b, f, g is an object $e \in Ob(calC)$ together with an arrow $u : e \rightarrow a$ s.t.*

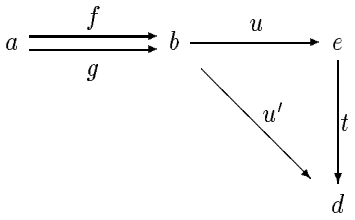
- $u : e \rightarrow a$, $f \circ u = g \circ u$
- $\forall d \in Ob(calC)$ *with* $u' : d \rightarrow a$,
  s.t. $f \circ u' = g \circ u'$,
  $!\exists t : d \rightarrow e$ *s.t.* $u \circ t = u'$



In ***Set****:* $e = \{x \in a | f(x) = g(x)\}$

Definition 3.25 *Coequalizer.*
Given calC a category, let $a, b \in Ob(calC)$ and $f, g : a \rightarrow b$, then the **coequalizer** *of a, b, f, g is an object $e \in Ob(calC)$ together with an arrow $u : b \rightarrow e$ s.t.*

- $u : b \rightarrow e$, $u \circ f = u \circ g$
- $\forall d \in Ob(calC)$ *with* u' : $e \rightarrow d$ *s.t.* $u' \circ f = u' \circ g$, $!\exists t : e \rightarrow d$ *s.t.* $t \circ u = u'$



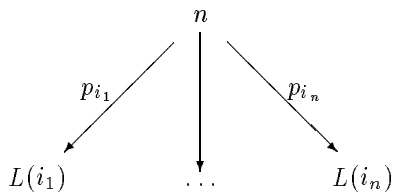
In ***Set****: Identifies objects in* b *having same preimage under* f, g

Sketches

Definition 3.26 *Linear Sketch [3].*
A linear sketch is a pair $(\mathcal{G}, \mathcal{D})$*, where* $\mathcal{G}$*, is a graph and* $\mathcal{D}$*, is a collection of diagrams in* $\mathcal{G}$*.*

Definition 3.27 *Model of Linear Sketch [3].*
A model of a linear sketch $\mathcal{S}$ *in a category* $\mathcal{C}$ *is a graph homomorphism* $M : \mathcal{G} \rightarrow \mathcal{C}$*, such that whenever* $D : \mathcal{I} \rightarrow \mathcal{G}$ *is a diagram in* $\mathcal{D}$*, then* $M \circ D$ *is a commutative diagram in* $\mathcal{C}$*.*

Definition 3.28 *Finite Discrete Cone [3].*
A finite discrete cone in a graph $\mathcal{G}$ *consists of a finite discrete graph* $\mathcal{H}$*, a graph homomorphism* $L : \mathcal{I} \rightarrow \mathcal{G}$*, a node* n *of* $\mathcal{G}$*, a collection of arrows* $pi : n \rightarrow L(i)$*, one for each node* $i \in \mathcal{I}$*. The node* n *is called the vertex of the cone, and the diagram* L *is called the base of the cone.*



Definition 3.29 *Finite Product Sketch [3].*
A finite product sketch or FP sketch $\mathcal{S}$ *is a triple* $(\mathcal{G}, \mathcal{D}, \mathcal{L})$*, where* $\mathcal{G}$ *is a finite graph,* $\mathcal{D}$ *a finite set of finite diagrams in* $\mathcal{G}$*,* $\mathcal{L}$ *a finite set of finite discrete cones in* $\mathcal{G}$*.*

Definition 3.30 *Model of Finite Product Sketch [3].*
Let $\mathcal{S} = (\mathcal{G}, \mathcal{D}, \mathcal{L})$ *be an FP sketch, and* $\mathcal{C}$ *be a category, a model of* $\mathcal{S}$ *in the category* $\mathcal{C}$ *is a model of the linear sketch* $(\mathcal{G}, \mathcal{D})$ *such that for any cone* $L : \mathcal{I} \rightarrow \mathcal{G}$ *in* $\mathcal{L}$*, the composite* $M \circ L$ *is a product cone.*

Particularly, the vertex n of the cone L is mapped to a product in category $\mathcal{C}$.

Definition 3.31 *Finite Discrete Sketch [3].*
A finite discrete sketch or FD sketch $\mathcal{S}$ *is a 4-uple* $(\mathcal{G}, \mathcal{D}, \mathcal{L}, \mathcal{H})$*, where* $\mathcal{G}$ *is a finite graph,* $\mathcal{D}$ *a finite set of finite diagrams in* $\mathcal{G}$ *and* $\mathcal{L}$ *a finite set of finite discrete cones in* $\mathcal{G}$ *with finite basis, and* $\mathcal{H}$ *a finite set of finite discrete cocones in* $\mathcal{G}$ *with finite basis.*

Definition 3.32 *Model of Finite Discrete Sketch [3].*
Let $\mathcal{S} = (\mathcal{G}, \mathcal{D}, \mathcal{L}, \mathcal{H})$ be an FD sketch, and $\mathcal{C}$ be a category, a model of $\mathcal{S}$ in the category $\mathcal{C}$ is a model of the linear sketch $(\mathcal{G}, \mathcal{D})$ such that any cone in L is mapped to a product cone, and any cocone in $\mathcal{H}$ is mapped to a sum cone.

Definition 3.33 *Finite Limit Sketch [3].*
A finite limit sketch or FL sketch $\mathcal{S}$ is a triple $(\mathcal{G}, \mathcal{D}, \mathcal{L})$, where $\mathcal{G}$ is a finite graph, $\mathcal{D}$ a finite set of finite diagrams in $\mathcal{G}$ and $\mathcal{L}$ a finite set of finite cones in $\mathcal{G}$.

Definition 3.34 *Model of Finite Limit Sketch [3].*
Let $\mathcal{S} = (\mathcal{G}, \mathcal{D}, \mathcal{L})$ be an FL sketch, and $\mathcal{C}$ be a category, a model of $\mathcal{S}$ in the category $\mathcal{C}$ is a model of the linear sketch $(\mathcal{G}, \mathcal{D})$ such that any cone in L is mapped to a limit cone.

Definition 3.35 *Sketch [3].*
A sketch $\mathcal{S}$ is a 4-uple $(\mathcal{G}, \mathcal{D}, \mathcal{L}, \mathcal{H})$, where $\mathcal{G}$ is a finite graph, $\mathcal{D}$ a finite set of finite diagrams in $\mathcal{G}$ and $\mathcal{L}$ a finite set of cones in $\mathcal{G}$, and $\mathcal{H}$ a finite set of cocones in $\mathcal{G}$.

Definition 3.36 *Model Sketch [3].*
Let $\mathcal{S} = (\mathcal{G}, \mathcal{D}, \mathcal{L}, \mathcal{H})$ be an sketch, and $\mathcal{C}$ be a category, a model of $\mathcal{S}$ in the category $\mathcal{C}$ is a graph homomorphism from $\mathcal{G}$ to $\mathcal{C}$ such that any diagram in $\mathcal{D}$ is mapped to a commutative diagram, any cone in $\mathcal{L}$ to a limit cone, and any cocone in $\mathcal{H}$ to a colimit cocone.

2-category

Definition 3.37 *2-Category [3].*
A 2-category $\mathbf{C}$ consists of three sets, C_0, C_1, C_2 (elements of C_i are called i-cells). The 2-category has a three category structures:

- *C^b, the base category, has C_0 as objects and C_1 as arrows*
- *C^h, the horizontal category, has C_0 as objects and C_2 as arrows*
- *C^v, the vertical category, has C_1 as objects and C_2 as arrows*
- *A 2-cell α goes between arrows whose vertical identities have the same horizontal domain and codomain, i.e. the following diagram holds:*

$$\mathcal{A} \underset{G}{\overset{F}{\Downarrow \alpha}} \mathcal{B}$$

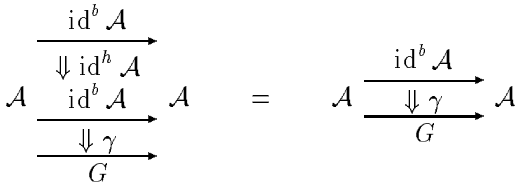
- *A 2-cell that is a horizontal identity must also be a vertical identity; i.e. for a 0-cell $\mathcal{A}$,*

$$\mathrm{id}^h\,\mathcal{A} = \mathrm{id}^v(\mathrm{dom}^v(\mathrm{id}^h\,\mathcal{A})) = \mathrm{id}^v(\mathrm{cod}^v(\mathrm{id}^h\,\mathcal{A}))$$

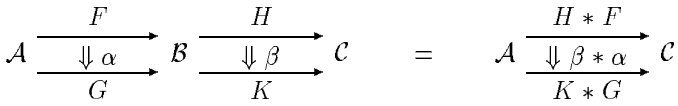
Thus for all 2-cells $\beta : H \to K : \mathcal{A} \to \mathcal{B}$,

$$\mathcal{A} \underset{\mathrm{id}^b\,\mathcal{A}}{\overset{\mathrm{id}^b\,\mathcal{A}}{\Downarrow \mathrm{id}^h\,\mathcal{A}}} \mathcal{A} \underset{K}{\overset{H}{\Downarrow \beta}} \mathcal{A} \quad = \quad \mathcal{A} \underset{K}{\overset{H}{\Downarrow \beta}} \mathcal{B}$$

For all $\gamma : \mathrm{id}^b\,\mathcal{A} \to G : \mathcal{A} \to \mathcal{B}$*:*



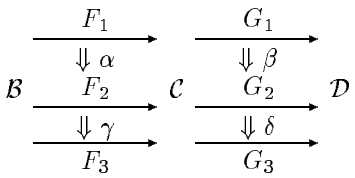
- *For 2-cells* α *and* β *with* $\mathrm{cod}^h\,\alpha = \mathrm{dom}^h\,\beta$*:*



- *For 2-cells* α*,* β*,* γ *and* δ *for which the composites* $\beta * \alpha$*,* $\delta * \gamma$*,* $\gamma \circ \alpha$*,* $\delta \circ \beta$ *are defined, then*

$$(\delta * \gamma) \circ (\beta * \alpha) = (\delta \circ \beta) * (\gamma \circ \alpha)$$

, i.e.



- *The structure of the base category is derived from those of the horizontal and vertical category by the following rules:*
  - *The domain and codomain of 1-cells are given by* $\mathrm{dom}^b(F) = \mathrm{dom}^h(\mathrm{id}^v(F))$ *and* $\mathrm{cod}^b(F) = \mathrm{cod}^h(\mathrm{id}^v(F))$
  - *The base identity* $\mathrm{id}^b(\mathcal{A})$ *for a 0-cell* $\mathcal{A}$ *is* $\mathrm{id}^b(\mathcal{A}) = \mathrm{dom}^v(\mathrm{id}^h(\mathcal{A})$*, which is the same as* $\mathrm{cod}^v(\mathrm{id}^h(\mathcal{A})$
  - *The composition is denoted* $*$ *and is defined by:* $G*F = \mathrm{dom}^v(\mathrm{id}^v(G)* \mathrm{id}^v(F))$

Cartesian Closed Categories

Definition 3.38 *Cartesian Closed Category (CCC) [3].*
A category $\mathcal{C}$ *is called a Cartesian closed category if it satisfies the following:*

- *There is a terminal object 1;*
- *Each pair of objects* A *and* B *of* $\mathcal{C}$ *has a product* $A \times B$ *with projections* $p_1 : A \times B \to A$ *and* $p_2 : A \times B \to B$*;*
- *For every pair of objects* A *and* B *there is an object* $[A \to B]$ *(called the exponential object) and an arrow eval:* $[A \to B] \times A \to B$ *with the property that for any arrow* $f : C \times A \to B$*, there is a unique arrow* $\lambda f : C \to [A \to B]$ *such that the composite:*

$$C \times A \xrightarrow{\lambda f \times A} [A \to B] \times A \xrightarrow{\mathrm{eval}} B$$

is f*.*

Toposes

Definition 3.39 *Subobject of a Category [3].*
A subobject of an object C in a category is an equivalence class of monomorphisms $C_0 \to C$ where $f_0 : C_0 \to C$ is equivalent to $f_1 : C_1 \to C$ iff there are arrows (isomorphisms) $g : C_0 \to C_1$ and $h : C_1 \to C_0$ such that $f_1 \circ g = f_0$ and $f_0 \circ h = f_1$.

Definition 3.40 *Subobject Functor [3].*
Given a category $\mathcal{C}$ having pullbacks, the subobject functor is a functor: $Sub : \mathcal{C}^{op} \to \mathbf{Set}$ such that:

- *For an object C, $Sub(C)$ is the set of subobjects of C*
- *For an arrow $k : C' \to C$, $Sub(k) : Sub(C) \to Sub(C')$*

Definition 3.41 *Topos [3].*
A topos is a category which (1) has finite limits, (2) is cartesian closed, (3) has a representable subobject functor.

A functor is representable iff it has a universal element.

Slice Category

Definition 3.42 *Slice Category [3].*
If $\mathcal{C}$ is a category and A any object of $\mathcal{C}$, the slice category $\mathcal{C}/A$ is defined by:

- *An object of $\mathcal{C}/A$ is an arrow $f : C \to A$ of $\mathcal{C}$ for some object C.*
- *An arrow of $\mathcal{C}/A$ from $f : C \to A$ to $f' : C' \to A$ is an arrow $h : C \to C'$ with the property that $f = f' \circ h$, we note this arrow $h : f \to f'$*
- *The composite of $h : f \to f'$ and $h' : f' \to f''$ is $h' \circ h$.*

Note that a slice category is a category.

The connected components

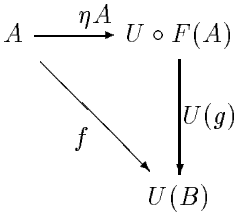
Definition 3.43 *Connected nodes [3].*
A node a can be connected to a node b of a graph $\mathcal{G}$ if it is possible to get from a to b following a sequence of arrows of $\mathcal{G}$ in either direction.

Definition 3.44 *Connected Components [3].*
"Being connected to" is an equivalence relation. An equivalence class of nodes with respect to this relation is called a connected component of the graph $\mathcal{G}$. If $\mathcal{G}$ is a category, a connected component of a category is also a full subcategory.

Adjunctions

Definition 3.45 *Left and Right Adjoints [3].*
Let $\mathcal{A}$ and $\mathcal{B}$ be categories. If $F : \mathcal{A} \to \mathcal{B}$ and $U : \mathcal{B} \to \mathcal{A}$ are functors, F is left adjoint to U and U is right adjoint to G provided there is a natural

transformation $\eta : id \rightarrow U \circ F$ *such that for any objects* A *of* $\mathcal{A}$ *and* B *of* $\mathcal{B}$ *and any arrow* $f : A \rightarrow U(B)$*, there is a unique arrow* $f : F(A) \rightarrow B$ *such that :*



4 Ideas for Formalization of Agents and MAS

Formalization helps understand what are the concepts we handle. As everybody has a different definition of agents, according to what he is doing with, it is worthless to formalize the "universal" agent. For this reason, we will define an agent much more like a process and extra primitives of agents will be investigated separately.

- *Computational units.*
  They are the atomic computation entities we can find in an agent. Depending on the level of abstraction, a computational unit can be: an instruction, a thread, a process, an agent. More basically, a computational unit is an action that an agent can take: a communication action, or the interpretation of a received message, a reasoning process, a calculus, etc.

- *Agents or Distributed Processes.*
  An agent is *composed* of several computational units disseminated in space and time; in space, because the computational units can run on different hosts or sites, in time, because the set of computational units changes in time due to the birth and death of the computational units, as well as to the change of internal state of the computational units.
  The big problem comes from *composition*. What does it mean here? Parallel execution of the units with or without communication?

  For intelligent agents, the internal state of such an agent comprises its knowledge, beliefs.

- *Services / Families of agents / Groups of agents.*
  A service is the *composition* of several agents disseminated in space and time. In space, because agents are not executing in the same place at the same time (if mobile or if made of computational units). In time, because the set of agents composing the service changes in time according to birth, death or change of state of the agents. Concerning the time, it is particularly interesting to consider agents which have a certain property. A service could be the staff of agents having this property but not necessarily at the same time. Here we have really the case of time dissemination.
  Here too *composition* has to be defined and it is not the same composition as those we find in the agents.
  Note that the definition of agent being more than fuzzy, services could also be called agents.
  Examples of services: (1) *Agent domains* of [13] are a collection of agents

associated with an agent name server. The goal is to to retrieve an agent given its name, in a similar way to packets in the Internet are routed according to hosts domains of IP. (2) *Messenger servers* of [19] are a collection of messengers (mobile code) which realize distributed services, e.g. mutex semaphore. (3) *Sessions* of [18] are collections of processes composed in parallel, and the bindings between the processes can change dynamically during a session.

- *Multi-Agent Systems (*MAS*) or Distributed Systems.*
  They are the *composition* of all the agents presents in all the places of the system and at any observable moment. Here *composition* only means the grouping in time-indexed sets of the agents. In a given set indexed by t we find all agents present at t in the system.

We have composition at three different levels: (1) the composition of computational units in order to realize an agent, (2) the composition of agents in order to perform a service, (3) a composition of services to realize a MAS or a whole distributed system. These three types of compositions induce three types of structure: a structure of agents, of services and of MAS.

Services allow us to examine the agents under different points of view. They can be the agents communicating together, or the agents which cooperate (even for a short time) to solve their goals, agents that share a special property, etc. We have different kinds of services: social and property based, and some of them will appear as *emergent*, i.e. they will be observed but not necessarily specified in advance.

Multi-agent systems are only the aggregate of the agents present at a given time, while services will be derived (emerged) from the whole multi-agent system.

Services seems to become a central issue in our discussion. The services are what we will use to structure agents. It is by the intermediary of the services that we will observe agents: how do they organize, compose, uncompose, etc. Thus, what we need is a tool which helps us detect services. We hope that category theory can be such a tool.

What about the intelligence? From our point of view, intelligence is an added feature, not present in all distributed systems. We think that all MAS are distributed systems, but not all distributed systems are MAS, thus we feel that intelligence can be included at the computational units level. Intelligent features are part of the internal state of an agent and computational units are responsible for modifying it.

5 Ideas for Category Theory applied to MAS

This section investigates several notions of category theory: it gives informally the meaning of the notion and how in the literature it has been used; it mentions for each of them how it seems to be related to agenthood described in the previous section. Our definitions or intuitive meaning of the following categorical notions come from [16, 3].

Category

A category is a collection of objects, together with a collection of arrows between these objects. Arrows follow some rules: arrows are composable, the composition operator is associative, and an identity arrow exists for each object. Objects and their arrows define a *structure*. Thus, a category defines objects and their structure. For example, think at the group structure: each object is a group and each arrow is a group homomorphism.

Two types of categories can be defined. Indeed, for groups, we can consider a *group as a category*, and we can consider the *category of all groups*. In the first case, there is only one object, and the arrows are all the elements of the considered group. In the second case, objects are groups and arrows are groups homomorphisms, i.e. mappings that preserve groups structure: a group is mapped to another group and its structure is mapped to the structure of the new group.

In the Petri Net community, several categories of Petri Nets have been defined: categories of petri nets with black tokens, of colored petri nets, of algebraic petri nets, etc. Classically, objects in the category are petri nets and arrows are mappings between petri nets that preserve the flow relations.

We will now list some ways to build categories for the definitions we have given in section 4.

- *Computational Units*
  - *Category of (all) Computational Units*
    Objects are information state: both internal information of agents and external information of agents. Internal information consists for example of the knowledge, beliefs, computation state of an agent, while external information consists of the information of the environment that the agent handles.
    Arrows represent the computational units, i.e. they stand for the change of information state. For example, a calculus, a reasoning, a speech act performative, a change of external information state.
- *Agents*
  - *An Agent as a Category*
    Objects are (external and internal) information state. Arrows are change of information state (computational units). The difference between an agent seen as a category and a category of computational units, is that in a category of computational units, the computational units can belong to one or more agents, while in an agent seen as a category, the computational unit (arrows) are concerned only with this agent.
  - *Category of Agents*
    Objects are agents and arrows are mappings between agents that preserve the agents' internal structure given by agents seen as categories. More precisely, objects are agents seen as categories and arrows are functors between these categories.

- *Services.*
  The structure of a service is the *composition* we have between the agents participating in the service. This composition depends on what service we are considering: agents realizing a given property, agents communicating information with each other, agents that cooperate, etc.

  - *A Service as a Category*
    Objects are agents, and arrows stand for the composition relation between the agents, i.e. arrows stand for either (1) relations between agents sharing a given property, or (2) communications between agents, coordination mechanisms etc.
  - *Category of Services*
    Objects are services and arrows are mappings between services preserving the services' structure. More precisely, objects are services seen as categories and arrows are functors between these categories.

- *Multi-Agent Systems*

  - *A* MAS *as a Category*
    A MAS can be seen as a category of services, thus objects are services and arrows are mappings preserving their structure. Another point of view consists in considering objects as services, and arrows as relations between these services, i.e. a communication between groups of agents, a collaboration between services, etc.
  - *Category of* MAS
    Objects are MAS and arrows are mapping preserving the MAS structure (of services, agents and computational units). More precisely, objects are MAS seen as categories and arrows are functors between these categories.
  - *Flat* MAS *seen as a category*
    A flat MAS consists in the aggregation of agents without structuring them into services. A flat MAS is made of agents executing in parallel. Objects are agents, and arrows are agents' state change. The composition of arrows can be seen as the sequential occurrence of two events.

Functors

Functors are functions between categories: objects and arrows of a category are mapped to objects and arrows in the target category respectively. Functors are transformations, from one category to another, preserving the internal structure of the categories.

Useful related definitions are: the forgetful (underlying) functor which forgets all or part of the structure induced by the category; full vs faithful functor, in the full functor, the induced mapping is surjective, in the faithful functor the induced mapping is injective.

In the Petri net community, once a category of petri nets has been defined, a semantics functor is given, which maps each petri net to a transition system or to a petri net of another category, and each arrow to an arrow between transition systems or petri nets.

Functors on the previous defined categories for agents:

- *Replacement*
  For agents seen as categories, a functor between such categories can be seen as the replacement of one agent by another one. It goes the same for services and MAS seen as categories, a functor between such categories can be seen as the replacement of one service or MAS by another one, respectively, where, for example, the coordinations mechanisms between agents and services are changed by the functor.
  It goes the same for functors between two categories of agents, two categories of services, two categories of MAS.

- *Membership*
  Functors, between a category of agents and a category of services, a category of services and a category of MAS, can be interpreted as a membership relation between agents and services, services and MAS.

- *Refinement*
  Functors, between a category of MAS and a category of services, a category of services and a category of agents, can be interpreted as an implementation of a MAS with services and of services with agents.

- *Semantics Functors*
  For computational units, agents, services and MAS, semantics functors between a category of computational units, agents, services, or MAS and a category of transitions systems, or petri nets is useful to express the semantics of the former categories.

Bifunctor

A bifunctor is a functor from a product category to a category. If the product category is considered as a category, the bifunctor is a functor, if the three involved categories are the same, the bifunctor can be considered as a binary operation on both objects and arrows.

Bifunctors [10] are used to model parallelism between agents. Two arrows in the single categories can be composed through the bifunctor in order to model the parallelization of the two events represented by the arrows.

Bifunctors can also be used for *composition*: in the three cases, composition of computational units, composition of agents and composition of services, the same mechanism of bifunctor can be applied. For computational units: a bifunctor where the three categories are an agent seen as a category, two computational units are mapped to a third computational unit which represents their composition. For agents: the underlying category is a service seen as a category, two agents are mapped with a third agent, which is their composition. It goes the same for services.

Natural Transformations

Natural transformations are functions between functors working on identical categories. Two different functors from a category $\mathcal{A}$ to a category $\mathcal{B}$, give two different transformations of the objects and structure of $\mathcal{A}$ into objects and

structure of $\mathcal{B}$. A natural transformation between these two functors gives in $\mathcal{B}$ a possible relation between the result of the two transformations given by the two functors.

Natural transformations can be seen as deformations of one construction into another, i.e. of the construction given by one functor into the construction given by the other functor.

A natural transformation can explain how two different semantics for an agent are related, how two different refinements can be considered equivalent, how two different coordination mechanisms are related, etc.

Commutative Diagrams

Commutative diagrams are used for expressing equations. Based on graphs, they express which arrows paths have to be equal in a category.

For algebraic specifications, commutative diagrams are useful for expressing which properties have to be realized by the operations defined in the algebraic specifications.

For isolated agents, commutative diagrams can be used to express properties of agents as for example: capable, perceptive, interpretive, etc. Similarly, for services and MAS, commutative diagrams can be used to express conditions or properties on services and MAS, e.g. the coherence of the global state of the MAS.

Products and Sums

Products and sums are a way of composing objects of a category. More generally, limits and colimits define the composition of objects. Given two objects of a category, their limit is a third object of this category together with two arrows (between this object and the other two), verifying some nice property.

Products are constructions allowing the definition of operations of arbitrary arity. Sums allow the specification of alternatives. For examples, in programming languages products are records with fields, in deductive systems products are conjunctions.

In Petri nets, nice categories of petri nets are those allowing the existence of colimits (coproducts, coequalizer, pushouts, etc): for each pair of petri nets in the category, their colimit exists. Usually, the fusion of nets is a coequalizer, the union of nets is a pushout.

More generally limits and colimits seems useful for representing *composition, coordination* of agents or groups of agents. We can consider isolated agents to be single objects, and services to be limits or colimits of these objects. Depending on the way agents compose, the service is either a pushout, a pullback, an equalizer, etc. Similarly, a MAS is the limit or colimit of services.

Sketches

Sketches are useful for specifying a structure, together with conditions on it (commutative diagrams), and for specifying the objects that are limits/colimits with some conditions. Indeed, a sketch is a graph together with some commutative diagrams. Sketches are used to express a structure where the graph gives

the operations and the diagrams give the equations to be satisfied (think at algebraic specifications in computer science).

Models of structure (sketch) are given by functors. Functors give models of this structure, and natural transformations are homomorphisms between the models.

Linear sketches allow unary operations, while linear sketches with constants allow nullary and unary operations. Finite Product (FP) sketches allow operations with more than one arguments. Models of an FP sketch can be taken in an arbitrary category with finite products. Finite Discrete (FD) sketches allow the description of objects which are sums or products Finite Limit (FL) sketches allow the use of finite limits (e.g. equalizers, pullbacks). General Sketches allow the use of limits and colimits.

Instead of defining a category of agents, and then verifying or not that this category allows coproducts or has some nice properties, sketches are useful for defining exactly what we want as a result for the category of agents: using sketches, we foresee what we expect to be limits and/or colimit, what conditions the category has to verify. A sketch alway induces a category, thus the induced category will have exactly the wanted properties.

2-category

A 2-category is a three categories structures. Three sets are used as objects and/or arrows in order to form three categories. The base category has the 0-cells as objects and the 1-cells as arrows which express the base structure. The 2-cells gives an extra structure involving all three types of cells. 2-cells are arrows in both the horizontal and the vertical category, thus they compose with two different composition operators, depending on the considered category (horizontal or vertical).

The most enlightening example of 2-category is given by the category of all categories, **Cat**. 0-cells are categories, 1-cells are functors, 2-cells are natural transformations. The base category has the categories as objects, and the functors as arrows, the vertical category has functors as objects, and natural transformations as arrows, the horizontal category has categories as objects and natural transformations as arrows.

The common use of 2-category is for defining a structure over another structure. 2-category is used for representing rewrite systems (rewrite rules are given by the vertical category), hence it can be used for refinement. The vertical category gives successive applications of rewrite rules or refinement operations, while the horizontal category gives the parallel application of rewrite rules or refinement operations.

2-category seems also useful for petri nets: the vertical category can be used for expressing the vertical composition of petri nets (refinement), and the horizontal category for the horizontal composition of petri nets (fusion, union of nets).

As 2-categories are used to represent a structure over another structure it seems natural to apply 2-categories to isolated agents and families of agents. Indeed, a 2-category gives several levels of abstraction, it seems similar to the several levels of abstraction we have with agents and groups of agents, groups of groups of agents, etc. There exists also the n-category, which can be used for structuring agents among several levels.

More precisely, for the 2-category, the base category is the category of isolated agents, the vertical category gives the families of agents and the horizontal category is the composition (in parallel) of families of agents.

Cartesian Closed Categories

A Cartesian Closed Category (CCC) is a category with a terminal object, all products, and for every pair of objects an exponential object exists ($[A \to B]$ is denoted B^A). The exponential object in the category of sets $B^{\{0,\cdots,n-1\}}$ is the set of all n-tuples of elements of B, or more generally B^A is the set of functions from A to B, and λf gives the currying of f. In other words, each arrow having a product as domain, can currified. Cartesian closed categories provide a theory which is equivalent to λ-calculus.

See the next subsection for the use of CCC for agents.

Toposes

Toposes are Cartesian closed categories (CCC) with an extra structure which proposes an object of sub-objects for each object. This extra structure makes toposes much more like the category of sets than other CCC. Toposes can be interpreted as categories of sets with internal system of truth values more general than the two-valued system of classical logic: an object in a topos can be seen as time-dependent set, or as a set with various degrees of membership (fuzzy logic). Toposes can be better for model computation than the category of sets (**Set**).

Toposes can help to express several features of agents:

- *Belief Degree*
  Toposes can serve to model a belief degree of an agent's belief.

- *Time Dependency*
  Several cases may occur: (1) objects are the state of the computational unit, agent, service, or MAS at different times; (2) an object in a topos represents an agent and all its computational units at different times, a service and all its inside agents at different times, a MAS and all its services at different times.
  To better understand we can make an analogy with fuzzy sets: consider a function $F : S \to \mathbb{N}$ from a set S to the set of all natural elements $\mathbb{N}$, if each element $s \in S$ stands for the state of a computational unit of a given agent, then $F(s)$ gives the time where this state occurs. The function F stands for the whole agent.

- *Space Dependency*
  An object in a topos represents the collection of all the disseminated computational units of an agent with their location. It goes the same for services: an object in a topos represent the collection of agents of the service with their location, and for MAS: an object in a topos represent a collection of services in a MAS with their location.
  The analogy with fuzzy sets is the following: consider $F : S \to \mathbb{N}$, where an element $s \in S$ stands for an agent in a given service and $F(s)$ is its location. The function F stands for the whole service.

- *Simultaneous Time and Space Dependency*
  The time-space problem is mentioned by Cap in [10]: "Processes and behaviors are graphs and thus define discrete geometric and topologic properties. It is a well-known problem in relativity theory and differential geometry that there is no adequate formalism to describe the dynamic change of topological properties of the space-time of general differential manifold as they might be caused by particle creation processes. (...) "
  Topos can help for the services. Services are groups of agents changing in space and time, due to the birth, death, change of state of agents. Toposes can help in following the different configuration services have in time, the set of the agents present at t in a service is different from the set of agents present at $t+1$ in the same service. Similarly for space-dependency, at time t the agents of the service are disseminated through a network according to a given configuration, at time $t+1$, these agents can have moved.
  With topos, we can envisage an object which represents an agent and whose internal representation is made of all computational units of this agent together with their location and time of occurrence.
  The analogy with fuzzy sets is the following: consider $F : S \rightarrow \mathbb{N} \times \mathbb{N}$, an element $s \in S$ stands for the state of an agent of a given service and $F(s) = (t, p)$ gives its position p at time t. The function F stands for the whole service.

Slice Category

"Think of objects of any slice category $\mathcal{C}/A$ as objects of $\mathcal{C}$ indexed by A" [3]. A could be a set of services, the agents are then indexed by the service they belong to. A functor from one slice category to another slice category with the same A represents a reorganization of the agents in the groups. If A changes, it represents a change of services.

The connected components

The connected components in a graph (a fortiori in a category) are arrows (paths) between the objects. In the case of categories, the connected components are full sub-categories of the category.

Components are respected under graph homomorphisms. The function which takes the component of object a to the component of object $f(a)$ (where f is a graph homomorphism) is a functor. The function which takes object a to its component is a natural transformation.

Connected components can also be useful to consider emergent properties or social emergence among agents. Indeed, services are groups of agents. The services will then exactly be given by the connected components.

6 Conclusion

Isolated agents are interesting for the AI community, it goes the same for multi-agent systems which are mostly investigated by the Distributed AI (DAI) community. The theories emerging from this community are primarily concerned

with intelligent properties of agents (intentions, etc) and communications among them.

However, poor work is undertaken on the distributivity of the system. No answers are given to questions like: what are the services available at a given time?, which hosts or locations are involved in a service? Migration, extension, diminution of a service.

Category theory provides an abstract mathematical tool to investigate computer science concepts. This working paper tries to apply category theory to the world of agents and multi-agents.

Category theory helps defining *structures*. We propose the notion of *service* as the basic structuring entity. A service is a collection of agents, that can be grouped together according to some common point, e.g. the agents of a service collaborate in order to realize a common goal, or the agents of a service all share a same property, etc. There can be an intersection between different services: a given agent can contribute to two different services. Agents appear (birth) and disappear (death) at any moment, so the service they are involved in increases or decreases. Agents can move, so a service can migrate or extend to one or multiple hosts.

This working paper informally gives some hints on how to apply some categorical notions to agents and especially to their composition. Composition is generally provided by limits and colimits, thus a service is a limit or colimit; time-space dependency of a service can be captured through toposes; and indexing of agents' sets can be realized with slice categories.

References

[1] *Symposium on Discrete Events and Manufacturing Systems, CESA96 IMACS Multiconference, IEEE-SMC*, 1996.

[2] O. Babaoglu and K. Marzullo. Consistent global states of distributed systems: Fundamental concepts and mechanisms. Technical Report UBLCS-93-1, Laboratory for Computer Science, University of Bologna, 1993.

[3] Michael Barr and Charles Wells. *Category Theory for Computing Science*. Prentice Hall International Series in Computer Science. Prentice Hall, second edition, 1995.

[4] E. Batard. L'anthropomorphisme en intelligence artificielle: une erreur de jeunesse ou problème mal posé? In *Troisièmes Journées Francophones sur l'Intelligence Artificielle Distribuée et les Systèmes Multi-Agents*, pages 387–396, 1995.

[5] Joseph Bates. The Role of Emotion in Believable Agents. *CACM*, 37(7):122–125, July 1994.

[6] R. A. Brooks. Intelligence without reason. In *IJCAI'91*, pages 569–595, 1991.

[7] R. A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.

[8] U. Hummert C. Dimitrovici. Composition of Algebraic High-Level Nets. In *Recent Trends in Data Type Specifications*, volume 534 of *LNCS*, pages 52–73. Springer-Verlag, 1990.

[9] C. H. Cap. An Operational Interpretation for Symmetric Monoidal Categories. Submitted to a conference.

[10] C. H. Cap. A Calculus of Distributed and Parallel Processes - An Approach Using Linear Logic and Algebraic Specification. Technical report, Department of Computer Science, University of Zürich, 1994.

[11] O. Etzioni and D. Weld. A Softbot-based Interface to the Internet. *CACM*, 37(7):72–76, July 1994.

[12] J. L. Fiadeiro. On the Emergence of Properties in Component-Based Systems. In M. Wirsing and M. Nivat, editors, *Algebraic Methodology and Software Technology*, number 1101 in LNCS, pages 421–443. Springer-Verlag, 1996.

[13] T. Finin, A. Potluri, C. Thirunavukkarasu, D. McKay, and R. McEntire. On Agent Domains, Agent Names and Proxy Agents. Technical report, Computer Science and Electrical Engineering, University of Maryland Baltimore County, 1996.

[14] R. Goodwin. Formalizing Properties of Agents. Technical Report CMU-CS-93-159, School of Computer Science, Carnegie Mellon University, 1993.

[15] J. Padberg H. Ehrig and L. Ribeiro. Algebraic petri nets petri nets revisited. In Springer-Verlag, editor, *Recent Trends in Data Type Specification*, LNCS 785, pages 188–206, 1992.

[16] S. Mac Lane. *Categories for the Working Mathematician.* Springer-Verlag New-York, 1971.

[17] J. Lilius. On the Compositionality and Analysis of Algebraic High-Level Nets. Technical Report 16, Digital Systems Laboratory, Helsinki University of Technology, 1991.

[18] K. Mani Chandy and A. Rifkin. Systematic composition of objects in distributed internet applications. In *30th Hawai International Conference on System Sciences.* IEEE, 1997.

[19] M. Muhugusa, G. Di Marzo, C. F. Tschudin, and J. Harms. Distributed Services in a Messenger Environment. Technical Report Cahier du CUI No 105, University of Geneva, 1996.

[20] Julia Padberg. An Outline of Rule-Based Refinement for Petri Nets. In *Formal Methods for Concurrency*, pages 16–22. Ludwig-Maximilians-Universität München, 1996.

[21] C. Piquemal-Baluard and S. Trouilhet. Déterminer l'état global d'une société d'agents: modèle et exemple. In *Troisièmes Journées Francophones sur l'Intelligence Artificielle Distribuée et les Systèmes Multi-Agents*, pages 219–230, 1995.

[22] David Pitt, David E. Rydeheard, and Peter Johnstone, editors. *Category theory and computer science: CTCS'95: proceedings*, number 953 in LNCS. Springer-Verlag, 1995.

[23] W. Reisig. Temporallogische Verifikation verteilter Algorithmen: Mehr als nur eine Variante. In *Formal Methods for Concurrency*, pages 27–30. Ludwig-Maximilians-Universität München, 1996.

[24] C. Sibertin-Blanc. Cooperative Nets. In Robert Valette, editor, *Application and Theory of Petri Nets 1994: proceedings*, volume 815 of *LNCS*, pages 471–490. Springer-Verlag, 1994.

[25] M. P. Singh. *A Theoretical Framework for Intentions, Know-How, and Communications.* Number 799 in LNAI. Springer-Verlag, 1994.

[26] Walter Van De Velde and John W. Perram, editors. *Agents breaking away: European workshop on modelling autonomous agents in a multi-agent world, MAAMAW'96: proceedings*, number 1038 in LNAI. Springer-Verlag, 1996.

[27] P. Wang. Comparing Categorization Models - A psychological experiment. Technical report, Center for Research on Concepts and Cognition, Indiana University, 1993.

[28] M. Wooldridge and N. R. Jennings. Agent Theories, Architectures, and Languages: A Survey. In Jennings Wooldridge, editor, *Intelligent Agents, ECAI-94, workshop on Agent theories, Architectures, and Languages (ATAL)*, number 890 in LNAI, pages 1–39. Springer-Verlag, August 1994.

[29] M. J. Wooldridge. *The Logical Modelling of Computational Multi-Agent Systems*. PhD thesis, Department of Computation, University of Manchester, 1992.

[30] Michael Wooldridge, Jörg P. Müller, and Milind Tambe, editors. *Intelligent agents II: agent theories, architectures, and languages: IJCAI'95 workshop (ATAL) proceedings*, number 1037 in LNAI, August 1995.

[31] Michael Wooldridgre and Nicholas R. Jennings, editors. *Intelligent Agents, ECAI-94, workshop on Agent theories, Architectures, and Languages (ATAL) proceedings*, number 890 in LNAI, August 1994.