Computational Semiotics : An Approach for the Study of Intelligent Systems Part I : Foundations

Ricardo Gudwin
DCA-FEEC-UNICAMP
gudwin@dca.fee.unicamp.br

Fernando Gomide
DCA-FEEC-UNICAMP
gomide@dca.fee.unicamp.br

ABSTRACT: The aim of this paper is to introduce the theoretical foundations of an approach for intelligent systems development. Derived from semiotics, a classic discipline in human sciences, the theory developed provides a mathematical framework for the concept of knowledge and for knowledge processing. As a result, a new perspective to study and to develop intelligent systems emerges. Here, Part I, the focus is on the essentials of such a theory. It includes the background needed from semiotics and describes the relationship between semiotics, intelligent systems and cognition. A taxonomy of the elementary types of knowledge is proposed and a classification of knowledge, from the point of view of application in cognitive systems, is addressed. In addition, a mathematical definition of objects and object networks, two key concepts within the computational semiotics paradigm proposed here, is introduced. The issues of knowledge modeling and system development, within the framework of computational semiotics, is the subject of a companion paper, Part II, where an illustrative example concerning control of an autonomous vehicle is included as well.

1. Introduction

Human intelligence has always been of interest and curiosity in the scientific world. First incursions in understanding and modeling intelligence and knowledge is rooted in Plato (Plato, 1991) and his philosophical speculations, being under constant investigation since then. With the arise of digital computers it became possible, in a sense, to turn from speculation to practice. Nowadays, many computer scientists and engineers believe that computers may show intelligent behavior. In the past, Turing pioneering study on the relation between computers and intelligence (Turing, 1950) raised interest and debates.

In the sixties, grounded mainly in mathematical and symbolic logic, the field of Artificial Intelligence (Nilsson, 1980) was born. During the 70's and 80's, many models for human knowledge were proposed. For instance, Newell introduced his physical symbol system (Newell, 1982) as a class of systems that embodies the essential nature of symbols. This would provide the necessary and sufficient condition for building generalized intelligent agents. Knowledge, to quote Newell, would be from a superior nature compared to ordinary computer programs, demanding a special treatment in a higher level, called the knowledge level.

One of the first knowledge representation models with repercussion was KL-ONE (Brachman & Schmolze, 1985). After came SOAR (Laird et.al. 1987; Norman, 1991), and later ACT* and PUPS (Anderson ,1989). All of these models aimed at architectures for a general intelligence. In 1989, Collins & Michalski (1989) introduced what should be a kernel for a theory of plausible inference. They called plausible inference any kind of inference pattern that could be exhibited by human beings.

In 1991, Albus published an outline for a theory of intelligence, and simultaneously Brooks (1991) argued that for an intelligent behavior, there should not necessarily exist representation or inference. Additional aspects of intelligence, e.g. approximate reasoning (including fuzzy or incomplete concepts), learning, prediction, and adaptation are being studied in the fields of computational intelligence (Zurada, 1994) and soft computing (Zadeh,1994). Considerable research effort on fuzzy set theory, neural networks and evolutive systems have and still are being pursued. The contribution of these fields in understanding the nature of human intelligence has been quite impressive.

Parallel to the developments in computer science and engineering, in human sciences there was a similar effort to model intelligence and intelligent behavior. Well known examples include the work of Piaget (Boden, 1983), and the development of semiotics by Peirce and Morris (Peirce, 1960; Morris, 1947; Morris, 1964; Morris, 1971), just to mention a few. Semiotics deals with signs (representations), objects (phenomena) and interpretants (knowledge), that is, the main issues in cognition and communication. Semiotics has shown to be an useful tool especially when the basic ingredients of intelligence and their relationships are of concern.

Despite the challenges in discovering the formal mysteries behind human intelligence, and the intrinsic difficulties in building machines and computer programs to emulate intelligent behavior, very few works analyzing intelligence in an integrated and organized manner have been done. Often, only particular aspects of intelligence are addressed. A notable exception comes from Albus' 1991 paper. In his work, Albus provides a systematic study of intelligence, and gives a description of the different parts composing the global phenomena. The integration of all parts should lead to intelligent behavior. Albus definitions and theorems are essentially linguistic due to the lack of a formal system to describe intelligence. In other words, currently there is no adequate mathematical model to describe intelligence as a whole. Most existing formalisms are closely tied to particular aspects, being unsuitable for a global formalization of intelligence. Semiotic Modeling and Situation Analysis-SSA, developed by Pospelov and his team in Russia was another important attempt in this direction. A key feature of the SSA approach is extraction of knowledge from the descriptive information by its consistent analysis based upon well established algorithms (Meystel and Albus, 1996). From this point of view, mathematical tools of semiotics are considered to include those used in control science, pattern recognition, neural networks, artificial intelligence, cybernetics. But semiotic specific mathematical tools (for combining signs, symbols and extracting meaning) are still in the process of development (Meystel and Albus, 1996).

In (Meystel,1996), the use of semiotics as a tool suitable for the analysis of intelligent systems was suggested. Concurrently, in (Gudwin,1996) the computational view of semiotics for modeling, development and implementation of intelligent systems, the computational semiotics approach, was proposed. Computational semiotics is build upon a mathematical description of concepts from classic semiotics. Its formal contents can be regarded as a contribution towards the development of semiotic specific mathematical tools. Thus, it is in the very realm of the formal foundations of intelligent systems. The main purpose of this paper is to introduce the mathematical aspects which subsumes computational semiotics. First, background concepts inherited from semiotics are briefly reviewed and two taxonomies for knowledge are proposed. In one taxonomy, the elementary hierarchy of signs, as originated by Peirce, is translated into a hierarchy of elementary types of knowledge. The second taxonomy, derived from the work of Morris, classifies knowledge according to its use by an intelligent system. After these preliminaries, the key concept of object is introduced and the corresponding mathematical characterization given. Next, the idea of object systems and object networks are formally developed. The use of those concepts to model each type of knowledge described here is left, although, to the companion paper, Part II, as well as an illustrative example concerning control of an autonomous vehicle.

2. Semiotics and Intelligent Systems

Intelligence has been a frequent item in human sciences agenda and, during the last decades, in the computer science and engineering area as well. In computer science and engineering, several and different aspects of intelligence have been modeled and used to develop computational systems with intelligent characteristics, notably in the area of artificial intelligence. (Barr & Feigenbaum, 1981a, Barr & Feigenbaum, 1981b, Cohen & Feigenbaum, 1981, Barr, Cohen & Feigenbaum, 1981).

In human sciences, intelligence and knowledge were and still are being studied and systematized within, *inter alia*, a discipline called semiotics. The idea of semiotics was introduced in the nineteen century by Charles Peirce in his work "Logic as Semiotics: The Theory of Signs" where the fundamentals of a theory of representation were proposed. See also Peirce (1960), Coelho Netto (1980) and Eco (1980). Essentially, semiotics studies the basic aspects of cognition and communication. Cognition means to deal with and to comprehend phenomena that occur in an environment. Communication means how a comprehended phenomena can be transmitted between intelligent beings. The basic unit of semiotics is called **sign**, defined as anything that in a certain way or aspect, represents something to someone (Peirce, 1960). Semiotics considers how signs are created, how they represent the different aspects of phenomenon, and how signs can be used to store and to transmit information.

Despite their common motivation, artificial intelligence and semiotics, as scientific disciplines, have followed different paths. Artificial intelligence has its roots in computer science and engineering and, as such, is based on sound theoretical and applied principles. Its goal is to provide structures and algorithms for complex problem solving. When it succeeds, the system derived is often said to be intelligent. To be successful, they may be able to emulate particular characteristics of intelligence. In contrast, semiotics has the goal of identifying, classifying and systematizing the different characteristics which, when put together, could be called intelligence. Semiotics is heavily rooted in the philosophical discourse and does not enjoy any theoretical or formal constructs as does AI. They also adopt different types of models. For instance, models used in artificial intelligence are either formal, structural, or symbolic whereas semiotic models are essentially descriptive, verbal. Therefore, intelligent behavior is more subjectively described in semiotics through concepts and labels without any immediate mathematical support. Most often semiotic models are intuitive and vague. Due to this difference, artificial intelligence models are more formally exact and well defined than semiotic models. On the other hand, semiotic models are deeper than AI models because they consider the multiple facets of intelligence. It is possible to relate models from each of these disciplines, but some semiotic models has no counterpart in AI models. Nevertheless, AI and semiotics do share a common characteristic: no systematic approach for an unified, integrated model of intelligence is provided by any of them. An adequate representation for such a model is beyond each individual scope, but their symbiosis seems to be a promise avenue to develop an unified view of intelligence. This is the main attempt of computational semiotics.

3. Cognition and Semiotics

In this section we discuss the concept of cognition. For this purpose, terms and ideas from semiotics are used to model the basic elements of cognition. Our discussion will, however, be centered in the computational interpretation of the underlying concepts.The reasons to focus the discussion in this perspective are twofold: first, it should help computer scientists and engineers to grasp the meaning of semiotics for computational purposes, and second to provide a computational interpretation of semiotics for the cognitive and information sciences field.

We start with a view about the nature of a cognitive system. For this purpose, consider the following scenario: a world (or environment) filled with objects. Let us assume that: a)- objects can be created and deleted, b)- objects are characterized by their attributes, c)- objects communicate (interact) with world trough input and output interfaces, and d)- objects can be modified.

A cognitive system is a system that: 1)-can identify world objects from its input (or input interface), and represent them by an internal structure, 2)-may detect changes in the object attributes, create new objects and destroy existing objects, representing these changes in its model, 3)-plan a modification in the world configuration from its internal model (e.g., changes in object attributes, creation of a new object, destruction of an object), and 4)-effectively act in the world (to implement planned modifications) trough its output interface,.

The input interface provides a way for a cognitive system to perceive world objects. In general the input interface gives only a partial view of the world. This is because the world perception may be restricted by, for example, the number, type and range of the sensors. Also, the complexity of the model used to represent the objects may vary, and only an approximate model is usually realistic. Note that a cognitive system is also an object in the world. Therefore, it may also extract knowledge about itself. The knowledge of itself as an object is under the same limitations as encountered in the representations of other world objects. This means that the result of the process of modeling world objects in internal representations may not be neither unique, nor exact. A cognitive system attempts to identify objects from its input interface data, but in principle there is no guarantee that the objects really do exist as such in the real world. In other words, although it is naturally prepared to identify objects, there is no explicit guarantee that the objects represented internally are actual world objects. A first basic element of a cognitive system responsible for judging when a world object is recognized is called an *interpretation*. An interpretation occurs when an internal representation of an object is evoked either from sensor data or from a previously identified internal representation. The source of information that originates the interpretation is called a *sign*, and the internal representation evoked is called the *interpretant*. For example, sensor data at an input interface can be a sign, but a previously evoked interpretant can also be a sign. Thus, the same sign at an input interface may generate a chain of interpretants associated with the multiple facets of a world object.

The triple (sign, object, interpretant) models the basic elements of cognition. It represents what is called a signic process, or *semiosis*, by Morris (Morris, 1971). Performing semiosis means extracting the meaning. Note some particularities of the model. For instance, the object, in the signic process, is not necessarily a physical object of the real world, but a presumable object of the real world. Usually the cognitive system does not have complete knowledge about the objects of the real world. Thus, the objects it recognizes can only be presumable, each of which represented by its interpretant. A common cognitive phenomenon, which helps to grasp this idea, is when a person, looking at the clouds in the sky, identifies objects that do not really exist. The human cognitive system presumes that there is an object in the contours of the clouds and creates an interpretant for it. Therefore, the real existence of an object is not a necessary condition for interpretation.

Another important issue about cognitive system is its static or dynamic characteristics. As an static cognitive system we may consider the case where a world model is assumed to exist within the system. Then, information coming from the input interface is compared to the existing model to generate actions that interact with world. The existing model remains unchanged. In contrast, a dynamic cognitive system modifies its model whenever the input data can not be identified with any object already known by the system. In this case, the system creates a new interpretant to represent the new object related with the input information. Therefore, we may state that dynamic cognitive systems are characterized by adaptation and learning. Assuming it is a dynamic system, we may assume a cognitive system being *initialized* with few (kernel) information about the world and let it continuously update its internal model. The model is enriched with new interpretants assigned to each object the system presumes to exist as it interacts with the environment. In practical terms, a dynamic cognitive system must have in its kernel a suitable structure to model world objects, and learning and adaptation methods for model update. The kernel provides the basic mechanisms for system auto-organization and evolution as well.

It is instructive to observe some existing relationships between (sign, object, interpretant) and the basic artificial intelligence constructs (representation, phenomenon, knowledge) as summarized in figure 1.

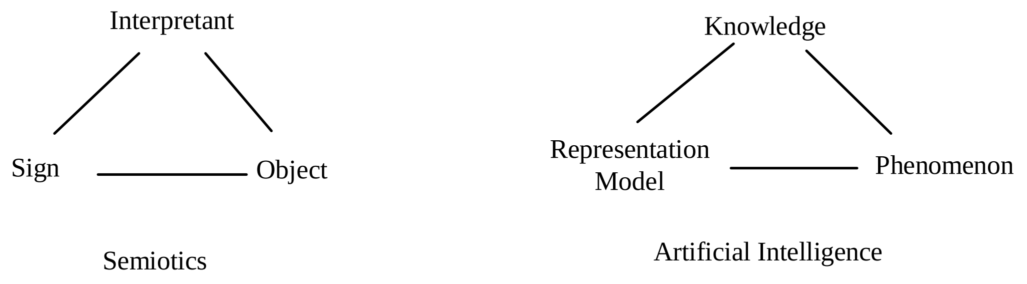

Figure 1 - Mapping Between Semiotics and Artificial Intelligence

In semiotics the interpretant is an intellectualization of the object whereas in AI a phenomenon is interpreted as knowledge. Knowledge is in AI associated with a representation model which in semiotics corresponds to a sign. In semiotics, the concept of object is not necessarily tied to physical objects. Abstractions of physical objects are the main kind of object. But the concept of object in semiotics also encapsulates actions, changes, truth, reasoning. An appropriate translation of the idea of object in semiotics is the idea of phenomenon in AI. Thus, the phenomenon in AI may be not only a physical object (or its abstractions), but may also refer to modifications in objects properties, or actions.

It should be pointed out that the mapping between (sign - object - interpretant) and (representation model - phenomenon - knowledge) is not one to one because the meaning of (representation model - phenomenon - knowledge) does not fully captures those of (sign - object - interpretant). For instance, in semiotics a given interpretant can be a sign for another interpretant, a fact not evident in AI models. When we say that a given interpretant is a sign for another, we are talking about two different natures of interpretants. In the first place, as an interpretant it corresponds to a semantics, an intellectualization of a phenomenon. Second, as a sign, it has a structural character which is essential to generate another interpretant. This dual nature of the interpretant can be a source of confusion when developing a formal model for (sign - object - interpretant). Actually, semiotics does not account for how a sign evokes an interpretant, and how this interpretant acts as a sign to generate another interpretant. Being descriptive, semiotic models simply consider the dual nature of interpretants natural. But, in formal and computational models, something must exist to convert a sign into an interpretant and, if it is the case, to convert this interpretant into a new sign. Therefore, the original triple (sign - object - interpretant) should be restructured before developing a formal model. However, we propose a particular structure to implicitly represent a particular knowledge. This structure may be transformed into a different structure to represent different knowledge. Hence the link between sign and interpretant (or representation model-knowledge) is preserved, with its meaning as advocated in semiotics. The point of an interpretant acting like a sign to generate a different interpretant can now be easily handled: given a structure (model), a transformation generates another structure (representing the new knowledge). The dual nature of the interpretant is now clearly represented. The transformation function is defined by a particular type of knowledge called argumentative knowledge.

4. Elementary Types of Knowledge

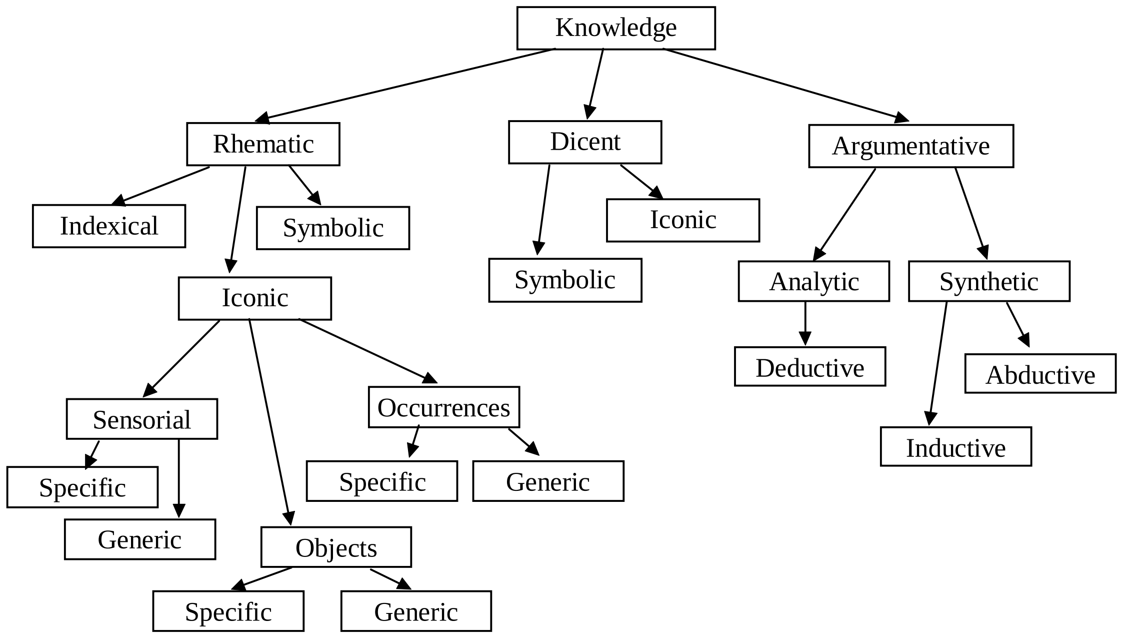

Figure 2 - Classification of Elementary Knowledge Types

In this section we address a classification of elementary knowledge types based on their basic nature. The presentation here is not intended to be exhaustive, but sufficient to cover a wide spectrum of knowledge types. Knowledge classification for applications in intelligent systems will be postponed until next section.

In the original work of Peirce, knowledge classification is found as a taxonomy of different types of signs (Peirce 1960). As we have seen, in semiotics signs are directly linked to their interpretants. Therefore, a taxonomy of signs induces a corresponding classification in the interpretants space. In AI models it is more convenient to work with a taxonomy of knowledge because for each type of knowledge a representation model must exist, but the representation model may be the same for different types of knowledge. Thus, it is more realistic classifying at knowledge level than at the representation model level. A classification of elementary knowledge types is summarized in figure 2 whose details are as follows.

4.1 Rhematic Knowledge

Rhematic knowledge is generated by the interpretation of rhemes or, equivalently, terms. Terms are used to refer to environment phenomena such as sensorial experiences, objects and occurrences.

There are three types of rhematic knowledge: symbolic, indexical, and iconic. The symbolic knowledge corresponds to a name. Using a name we can refer to an environment phenomenon.

The indexical knowledge is also used for referring an environment phenomenon, but not in an absolute way as in the symbolic case. Reference to a phenomenon is relative, and starts from a previously identified phenomenon. Examples include relative pronouns (this, these, that, etc.) in natural languages, indexes to identify terms in an ordered list in computations, or a coordinate system for a mobile referential.

The iconic knowledge corresponds to a direct model of the phenomenon it represents. It can be splitted into three basic categories: sensorial, object, and occurrence knowledge. Each of these can be specific or generic.

Sensorial knowledge refers to information that can be mapped directly to environment, through special devices (sensors and actuators). In the case of sensors, this information is collected by the devices and translated to a sensorial knowledge. In the case of actuators, a sensorial knowledge is used to change environment through the device. Specific sensorial knowledge concerns the information relative to a particular temporal instance, i.e., a specific state of sensors or actuators. Generic sensorial knowledge refer to a set (eventually infinite) of specific sensorial knowledges, usually sharing common characteristics. In this case, it can be defined by the properties that allow a specific knowledge to belong to this set. Sometimes, to define this set, one specific knowledge is chosen as a "prototype", and all other specific knowledges that are close to the prototype are considered to be in the set. In this case, we say that the prototype "represents" the generic sensorial knowledge. Many knowledge processing systems use only sensorial knowledge. Examples include artificial neural networks (Kosko, 1992), where input and output are specific sensorial knowledges and fuzzy control systems (Kosko, 1992; Pedrycz, 1989), where input is a specific sensorial knowledge, that is compared to generic sensorial knowledges given by fuzzy sets, generating other generic sensorial knowledges that, after defuzzyification, will generate a specific sensorial knowledge – the output. In this sense, fuzzy sets are a way of representing generic sensorial knowledge. Note that sensorial knowledge can be from a multi-dimensional nature, and multi-temporal as well.

The object knowledge builds upon sensorial knowledge; it assumes the existence of an object from which sensorial patterns are produced. Object knowledge can be specific or generic. Specific object knowledge means focusing at a particular object. This implies the existence of a model for such an object, and a structure suitable for representing its attributes in different instances of time. In other words, the object has a specific time trajectory: it is generated at a time, it exists during a time interval, and eventually it is destroyed at a later time. Generic object means a set of specific objects sharing similar characteristics. A simple example should help to distinguish specific and generic objects. When reading a page of a book suppose that the first letter is the letter "a". This is a specific object: the first letter "a" of the page. As we continue to read we may find many others letters "a", but if we refer to any of them it is meant a generic object. The property of being specific or generic does not depend on the particular time trajectory of each object, but on the identification of the structure that keeps the information about the object. Each specific object has an individuality assured by the associated model. A system may not have complete information (in all its temporal manifestations) about the object, but this does not avoid it to know that such an object exists. Generic objects are abstractions of specific objects, and cannot exist without the definition of the specific objects. Like generic sensorial knowledge, a generic object knowledge corresponds to a set of specific objects. In the same way, this set can be defined by a set of rules or characteristics that determine if a specific object can be associated with a generic object, or by means a "prototype".

Occurrence knowledge (or actions) means: to know the attribute values of world objects in a given time, any change of these values during time, and the generation or destruction of objects. Specific occurrence knowledge is linked to an occurrence related to one or more objects, at a particular instant of time (or at specific time steps). Generic occurrence knowledge has a similar nature, but it does not specify the time instants or time steps. Note that a)-an occurrence should always be related to one or more object or sensation, and b)-what characterizes a specific or generic occurrence is the definition of the time steps, not if the objects (or sensations) related are specific or generic. As an example to clarify these points, consider an autonomous vehicle within an environment. The vehicle may start moving from an initial position, go to several locations during, say, a day and stop at a final location. Suppose a fleet of 5 vehicles. A specific occurrence knowledge with a specific object could be: vehicle 1 started at 8:00 at location A, approached location B at 8:03, stayed there until 8:05, went to location C arriving at 8:07. A specific occurrence with a generic object could be: some vehicle (not specified) started at 8:00 at location A, approached location B at 8:03, stayed there until 8:05, went to location C arriving at 8:07. A generic occurrence knowledge with a specific object could be: vehicle 1 started at location A, went to location B and after to location C. Note that in this case we do not specify the time instants. In principle, if the vehicle run trough the same trajectory many times during the day, the generic occurrence can be related to any of them. A generic occurrence with a generic object could be: some vehicle started at location A, went to B and after to C. A more generic occurrence knowledge would be: some vehicle followed a trajectory, and reached three different locations.

Occurrence knowledge is with no doubt the most complex rhematic knowledge. Sensorial knowledge is primitive whereas object knowledge has a more abstract nature when explaining sensorial knowledge. Occurrence knowledge deals with presumable objects and abstracts about changes of the object attributes. Sensorial knowledge is independent of other types of knowledge. Object knowledge may be more sophisticated, e.g. when an object is a composition of other objects (called its parts), the hierarchy of parts being as complex as needed. Occurrence knowledge always depends on other types of knowledges, because it needs them to be meaningful. In general, to have an occurrence an object (or a sensorial knowledge, embodied into an object) should always exist. There may be more than one object involved in an occurrence. For example, a vehicle may transport a tool.

4.2 Dicent Knowledge

Dicent knowledge is characterized by truth values linking sensorial, object or occurrence knowledge with world entities. Recall that in classic symbolic logic (Chang & Lee, 1973) truth values are either 0 or 1 whereas in multivalued logic truth values are typically within the [0,1] interval.

The terms forming an expression can be either primitive propositions or logic connectives. There are two basic types of primitive propositions: iconic propositions and symbolic propositions. An iconic proposition is, by definition, a composition of terms forming a sentence. Each term refers directly or indirectly to an iconic rhematic knowledge. The rhematic knowledge is directly responsible for the truth value of a sentence. Each sentence must have at least one occurrence knowledge and one or more objects and/or sensorial knowledge. The truth value associated with a sentence models the degree of belief a cognitive system has about knowledge within iconic proposition. In other words, effectively associated with what actually happened with world objects.

In iconic propositions, the terms related to occurrence knowledge are called verbs (or predicates), and each related object called a *relatum* (or subject). The number of relata needed to express the meaning of each occurrence is its arity.

Symbolic propositions are names associated with other propositions. Their truth value should agree with the truth value of the proposition it names. Although it does not have any immediate rhematic knowledge associated, its truth value can be found from the interactions among other propositions.

A proposition is called a simple proposition if it can not be decomposed into more than one proposition, i.e., it is uniquely formed by a primitive proposition. A composed proposition is a proposition where different primitive propositions are linked by logic connectives.

A particular proposition of interest is the conditional proposition. A conditional proposition is expressed by

if (antecedent proposition) then (consequent proposition)

and clearly is a compound proposition. If a conditional proposition is true, from the truth value of the antecedent proposition one can determine the truth value of the consequent proposition. A proposition with truth value equal to 1 is called a fact.

Dicent knowledge in the form of symbolic propositions has been widely used in classic symbolic logic because it does not need all the semantic details behind the rhematic knowledge of iconic propositions. We assume that the truth value of a set of symbolic propositions are given, and by using conditional propositions and the “modus ponens” law of inference, we find the truth values of the remaining propositions.

When both symbolic and iconic propositions are used, dicent knowledge has a wider scope than does classic logic. For instance, in classic logic we often assume given the truth value of some propositions and from these we determine the truth values of other propositions. Using iconic propositions there is no need to assume truth values as given. The truth values of iconic propositions can be derived from its rhematic knowledge. Once the truth values of the rhematic components are known, one can find the truth value of other propositions (either symbolic or iconic).

4.3 Argumentative Knowledge

Argumentative knowledge brings the idea of argument as a transformation agent. An argument typically transforms premises (existing knowledge) into conclusions (new knowledge). The transformation is performed by argumentative function. Argumentative functions are transformation functions characterizing the argument. The arguments can be either analytic or synthetic.

Analytic arguments have its conclusion implicit in the premise. The task of the argumentative function in this case is to express the conclusion explicitly from the analysis of the premise. Thus, conclusions derived from analytic arguments are never in contradiction with existing knowledge in a cognitive system because they are just an expression of existing knowledge. Analytic arguments are also known as deductive arguments. A well known example of analytic argumentative function is the *modus ponens* (Enderton, 1972).

Synthetic arguments produces new conclusion from its premise. Conclusions derived from synthetic arguments may be contradictory with existing knowledge. Synthetic arguments allows knowledge discovery and refinement, an important requirement for learning. They can be classified as inductive or abductive. An inductive argument uses the knowledge in the premises to generate new knowledge as a conclusion, in the following way. The argument aims at modifying slightly the knowledge in the premise in the hope it will be kept valid. Inductive arguments support generalizations. That is, after observing a certain number individual occurrences, the inductive argument states that these occurrences are due to general rules. For example (Peirce, 1960), assume we took a brown bean from a box. If we take a second, a third, a fourth, etc, and they are all brown beans we may conclude than that all beans in the box are brown. This conclusion is not a deduction because to be the case, we should check all beans within the box. In the example, the inductive argument is the mechanism of believing in the truth of the proposition stated (conclusion) without complete information to warrant its truth. The basis of inductive arguments relies on the assumption that if a situation is frequently observed, then there exists a rule characterizing the situation. Inductive arguments may, as stated previously, introduce contradictions. In the example above, if only one bean were black, the proposition “all beans are brown” would be false (in the classic logic sense). Nevertheless, inductive arguments are important because there are many cases where checking all possible situations is not feasible.

The abductive argument uses the premise to refute candidate conclusions. For this purpose, a knowledge generator (eventually random) is assumed to exist. The abductive argument validates each knowledge generated verifying if it is not in contradiction with the premise. The basis of abductive arguments relies on assuming a piece of knowledge as valid if it does not contradict existing knowledge. In other words, the conclusion of an abductive argument is assumed to be valid because it agrees with current, existing knowledge. An example (also from Peirce) of abductive argument is the equation of planets movement derived by Kepler. He concluded that the equation holds because it was in agreement with his measurements. But, there is no proof that his equation is the ultimate equation. However, the idea behind abductive arguments is that if a conclusion is false, then sooner or later an evidence to refute the current conclusion will be found and the current knowledge will be corrected. Figure 3 summarizes the type of arguments discussed.

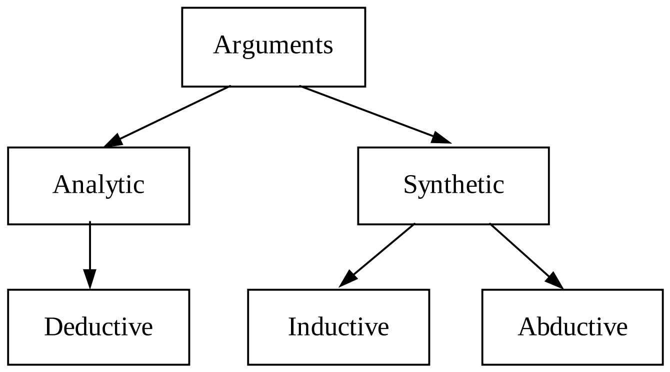

Figure 3 - Classification of Arguments

Inductive and abductive arguments may work in close cooperation. The result is a sort of learning scheme because inductive arguments may be used to generate new knowledge and abductive arguments to validate them.

In principle, there is no constraint on the type of knowledge within premises and conclusions. They can be rhematic, dicent or argumentative. Therefore, an argumentative knowledge may generate different argumentative knowledge, modify existing argumentative knowledge, and eventually destroy those argumentative knowledge which are, at certain point, considered wrong.

Deductive, inductive and abductive arguments are implicitly used in all knowledge based systems. Inductive and abductive arguments are used whenever the system involves some sort of learning. For example, fuzzy production systems (Yager & Zadeh, 1992) are instances of deductive argument. Neural networks (Kosko, 1992) uses deductive argument when used (after learning or programming) and inductive argument when there is learning involved. Genetic algorithms (Goldberg, 1989), uses inductive arguments when performing crossover or mutation, and abductive arguments when doing selection. Classifier systems (Booker, et.al. 1989) use deductive arguments in the classifier, and inductive and abductive arguments in the genetic algorithm (bucket brigade algorithm) part.

5. Applied Knowledge

In this section we discuss knowledge from the point of view of their use in cognitive systems. Figure 4 provides an overview of the classes of applied knowledge to be detailed next.

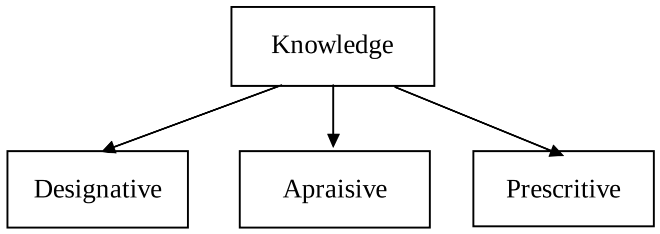

Figure 4 - Classification of Applied Knowledge

Based on its intended use, knowledge can be classified as designative, apraisive or prescriptive, terms coined by Morris (Morris, 1971; Morris, 1964; Morris, 1947). This classification is complementary to the elementary types of knowledge. In principle, any elementary type of knowledge can be used as designative, apraisive or prescriptive although some elementary types are more suitable to represent some types of applied knowledge (e.g. sensorial knowledge to represent apraisive knowledge).

5.1 Designative Knowledge

Designative knowledge models the world. For this purpose it uses rhematic, dicent and argumentative knowledge, either specific or generic. Designative knowledge can also be viewed as descriptive knowledge. A cognitive system initially has just a few, or eventually no designative knowledge at all. Usually designative knowledge emerges from the interaction between the system and world.

5.2 Apraisive Knowledge

Apraisive knowledge is a type of knowledge used as an evaluation, a judgment, a criteria to measure the success in achieving goals. In natural systems, apraisive knowledge is closely related with the essential goals of a being; reproduction, survival of the individual, survival of the specie, increasing knowledge about the world, for example. Depending on the goal it assumes special forms like: desire, repulse, fear, anger, hate, love, pleasure, pain, confort, disconfort, etc. Essentially, apraisive knowledge evaluates if a given sensation, object, or occurrence is good or not, as far as goal achievement is concerned.

Apraisive knowledge is usually a basic instinct, a native characteristic that expresses the goals of cognitive systems. In living beings for example, these goals includes growth, multiplication, etc., but in artificial systems apraisive knowledge is specified to reach a desired behavior..

5.3 Prescriptive Knowledge

Prescriptive knowledge is intended to act on the world. Basically, prescriptive knowledge is used to establish and to implement plans through actuators. However, prescriptive knowledge will not necessarily end up with an action. Prescriptive knowledge may also be used to do predictions, but only one of them is selected to generate an action.

To be called prescriptive, a knowledge should have some connection to the rhematic knowledge associated to actuator values. This connection can be direct or hierarchically structured.

A typical example of prescriptive knowledge is a command. The command can eventually be decomposed into subcommands until the level of actuators is achieved to be executed. Observe that a high level command may impose a latency time to be executed. This can eventually cause synchronization problems.

6. Foundations for a Mathematical Theory of Objects

This section introduces concepts and definitions as a formal background for computational semiotics. The focus here is on the main issues and definitions only. For a more in depth coverage the reader is referred to (Gudwin, 1996). The definitions assume a discrete set N, associated with time instants or algorithm steps when convenient. Extensions to the continuous case may be possible, but it will not be considered here. Usually N is the set of natural numbers, but in general it can be any countable set.

A remark concerning notation: no distinction between a function and its graph will be made. Therefore, both $f : A \to B$ and $f \subset A \times B$ will be used to express the function f.

DEFINITION 1 - Tuples

Let $q_1 , q_2 , \dots , q_n$ be generic elements of the sets $Q_1 , Q_2 , \dots , Q_n$ respectively. A tuple is a structure joining $q_1 , q_2 , \dots , q_n$ into a single element denoted by $q = (q_1 , q_2 , \dots , q_n)$.

A tuple with n elements is a n-tuple, or tuple for short. The elements of a tuple are called components. They can be identified by the indices associated with the order they appear. Note that a tuple component can be itself a tuple. In this case they are called complex tuples. For example $q = (q_1 , (q_{21} , q_{22} , q_{23}), q_3 , q_4 , q_5)$ is a complex tuple. To simplify notation we may assign $q_2 = (q_{21} , q_{22} , q_{23})$ turning the original tuple into $q = (q_1 , q_2 , q_3 , q_4 , q_5)$.

DEFINITION 2 - Tuple Arity

Let $q = (q_1 , q_2 , \dots , q_n)$ be a tuple. The **arity** of q, Ar(q), is the number of components $Ar(q) = n$.

Examples: $q = (a,b,c)$, $Ar(q) = 3$; $q = (a,(b,c),d)$, $Ar(q) = 3$; $q = ((a,b,c),(d,(e,f),g))$, $Ar(q) = 2$.

DEFINITION 3 - Reference Index

Let q be tuple. To identify an individual component of q we associate a reference index to each of its element. For a simple tuple the reference index will be a number i, $1 \leq i \leq Ar(q)$. For a complex tuple the reference index will be a simple tuple i with each element i_k corresponding to a sub-index of level k. The value of the sub-indices ranges between one and the arity of the tuple at level k. The reference index can also be used to identify the domain of the components. For example, let $s \in S$ be simple, and $c \in C$ complex tuples. Thus the reference indices and the component domains are found as follows:

$s = (a,b,c)$, $S = S_A \times S_B \times S_C$
$i=1 \to s_i = a$, $S_i = S_A$
$i=2 \to s_i = b$, $S_i = S_B$
$i=3 \to s_i = c$, $S_i = S_C$
$c = (a,(b,d))$, $C = C_A \times (C_B \times C_C)$
$i=1 \to c_i = a$, $C_i = C_A$
$i=2 \to c_i =(b,d)$, $C_i = C_B \times C_C$

$i=(2,1) \to c_i = b$, $C_i = C_B$
$i=(2,2) \to c_i = d$, $C_i = C_C$
$c = (a,(b,(s,d,(e,f),g),h))$, $C = C_A \times (C_B \times (C_C \times C_D \times (C_E \times C_F) \times C_G) \times C_H)$
$i=(2,1) \to c_i = b$, $C_i = C_B$
$i=(2,2,3) \to c_i = (e,f)$, $C_i = C_E \times C_F$
$i=(2,2,3,2) \to c_i = f$, $C_i = C_F$
$i=(2,3) \to c_i = h$, $C_i = C_H$
$i=2 \to c_i = (b,(s,d,(e,f),g),h)$, $C_i = C_B \times (C_C \times C_D \times (C_E \times C_F) \times C_G) \times C_H$

DEFINITION 4 - Induction Formula

Let $q = (q_1, q_2, \ldots, q_n)$ be a tuple and k be an expression defined by the following syntax

$k \leftarrow [\, i \,]$
$i \leftarrow i\, ,\, i$
$i \leftarrow [\, i\, ,\, i \,]$

where i is a reference index of q. The expression k is called an induction formula.

Examples: $k = [\, i_1, [\, i_2, i_3, i_4 \,], i_5 \,]$
$k = [\, [i_1, i_2], [i_3, [i_4, i_5]\,]\,]$
$k = [i_1, i_2, i_3]$
where i_j are reference indices of q.

DEFINITION 5 - Induction of a tuple

Let $q = (q_1, q_2, \ldots, q_n)$ be a tuple in $Q = Q_1 \times \ldots \times Q_n$ and k be an induction formula. The induction of q according k is defined as the new tuple $q_{(k)}$ induced by the induction formula. The induced tuple $q_{(k)}$ is found from k by changing brackets and each reference index i_j into parenthesis and q_{i_j} of the original tuple q. The domain $Q_{(k)}$ of $q_{(k)}$ is found similarly.

Examples: $q = (a,b,c,d)$, $Q = Q_1 \times Q_2 \times Q_3 \times Q_4$, $k = [1,3,4,2]$,
$q_{(k)} = (a,c,d,b)$, $Q_{(k)} = Q_1 \times Q_3 \times Q_4 \times Q_2$

$q = (a,b,c,d)$, $Q = Q_1 \times Q_2 \times Q_3 \times Q_4$, $k = [4,1]$,
$q_{(k)} = (d,a)$, $Q_{(k)} = Q_4 \times Q_1$

$q = (a,b,c,d)$, $k = [\,1, [2, 3]\,, 4]$,
$q_{(k)} = (a, (b,c), d)$, $Q_{(k)} = Q_1 \times (Q_2 \times Q_3) \times Q_4$

$q = (a,(b,c),d)$, $Q = Q_1 \times (Q_2 \times Q_3) \times Q_4$, $k = [1,(2,1),(2,2),3]$,
$q_{(k)} = (a,b,c,d)$, $Q_{(k)} = Q_1 \times Q_2 \times Q_3 \times Q_4$

$q = (a, (b,c), d)$, $Q = Q_1 \times (Q_2 \times Q_3) \times Q_4$, $k = [3,2]$,
$q_{(k)} = (d,(b,c))$, $Q_{(k)} = Q_4 \times (Q_2 \times Q_3)$

$q = (a, (b,c), d)$, $Q = Q_1 \times (Q_2 \times Q_3) \times Q_4$, $k = [3,2,(2,1)]$,
$q_{(k)} = (d,(b,c),b)$, $Q_{(k)} = Q_4 \times (Q_2 \times Q_3) \times Q_2$

DEFINITION 6 - Sub-tuple

A tuple $q_{(k)}$ is called a sub-tuple of q if k has only one pair of brackets, and each reference index in k is unary and appears only once in k.

DEFINITION 7 - Relation

If $R_1, \ldots, R_n$ are sets and $R = \{ (r_{i1}, \ldots, r_{in}) \}$, $i = 1, \ldots, M$, is a set of M tuples with arity $n > 1$ such that $\forall i \in \{1, \ldots, M\}$, $\forall k \in \{1, \ldots, n\}$, $r_{ik} \in R_k$, then the set R, $R \subseteq R_1 \times \ldots \times R_n$ is a relation in $R_1 \times \ldots \times R_n$,

DEFINITION 8 - Projection

Let $R = \{r_i\}$, $r_i = (r_{i1}, \ldots, r_{in})$ be an n-ary relation in $R_1 \times \ldots \times R_n$ and k be an induction formula with unary indices $k = [k_1, k_2, \ldots, k_m]$, $k_i \in \{1, \ldots, n\}$, $k_i \neq k_j$, if $i \neq j$, $i = 1, \ldots, m$, $j = 1, \ldots, m$, $m \leq n$. The projection of R on $R_{k_1} \times \ldots \times R_{k_m}$, denoted by $R \downarrow R_{k_1} \times \ldots \times R_{k_m}$ (alternatively, $R_{(k)}$) is the relation obtained by the union of all sub-tuples $r_{i(k)} = (r_{ik_1}, \ldots, r_{ik_m})$ of R originated from the induction of R's tuples according to k, $R_{(k)} = \cup\, r_{i(k)}$.

Examples: $A = \{1, 2\}$ $B = \{a,b,c\}$ $C = \{\alpha, \beta, \gamma\}$. $R=\{(1,a,\beta), (2,c,\alpha), (2,b,\beta), (2,c,\beta)\}$
$R \downarrow A \times C = \{ (1,\beta), (2,\alpha), (2, \beta) \}$
$R \downarrow C \times B = \{ (\beta,a), (\alpha,c), (\beta,b), (\beta,c)\}$

DEFINITION 9 - Free Projection

Let $R = \{r_i\}$, $r_i = (r_{i1}, \ldots, r_{in})$ be an n-ary relation defined in $U = R_1 \times \ldots \times R_n$ and k be an induction formula. The free projection of R in $U_{(k)}$, $R \downarrow U_{(k)}$ (alternatively, $R_{(k)}$) is the relation obtained by the union of all sub-tuples $r_{i(k)}$ originated by the induction of the tuples from R according to k $R_{(k)} = \cup\, r_{i(k)}$.

NOTE: Free projection is a generalization of projection. Recall that in a projection, the induction formula has unary indices only. This implies in tuples defined only over the main dimensions of the original tuple. In free projection, any element, in whatever level of a tuple, can be used to define the inducted tuple. Clearly, with the proper induction formula free projection becomes standard projection.

DEFINITION 10 - Cylindrical Extension

Let $R = \{ (r_{i1}, r_{i2}, \ldots, r_{in}) \}$ be an n-ary relation in $R_1 \times \ldots \times R_n$. The cylindrical extension P of R in $P_1 \times \ldots \times P_m$, denoted by $P = R \uparrow P_1 \times \ldots \times P_m$, where $\forall k \in \{1, \ldots, n\}\ \exists P_j = R_k$, $1 \leq j \leq m$, is the greatest (in the sense of the greatest number of elements) relation $P \subseteq P_1 \times \ldots \times P_m$ such that $P \downarrow R_1 \times \ldots \times R_n = R$.

Example: $A = \{1, 2\}$ $B = \{a,b,c\}$ $C = \{\alpha, \beta, \gamma\}$. $R = \{ (1,a), (2,c) \}$
$R \uparrow A \times B \times C = \{ (1,a,\alpha), (2,c,\alpha), (1,a,\beta), (2,c,\beta), (1,a,\gamma), (2,c,\gamma) \}$
$R \uparrow C \times A \times B = \{ (\alpha,1,a), (\alpha,2,c), (\beta,1,a), (\beta,2,c,), (\gamma,1,a,), (\gamma,2,c,)$.

NOTE: As in projection, the order of elements in tuples of the cylindrical extension it is not the same as in the original tuples.

DEFINITION 11 - Junction

Let R and S be two relations in $R_1 \times \ldots \times R_n$ and $S_1 \times \ldots \times S_m$, respectively, and $P = P_1 \times \ldots \times P_o$ an universe where $\forall i \in \{1, \ldots, n\}\ \exists P_k = R_i$, and $\forall j \in \{1, \ldots, m\}\ \exists P_h = S_j$, $o \leq n + m$. The junction of R and S under P, denoted by $R * S |_P$, is $R * S |_P = R \uparrow P \cap S \uparrow P$.

NOTE: If there is a $R_i = S_j$ then there may be only one set P_k with elements in tuples of R*S. In this case, for the tuples to be included in junction, the value of such element in the tuple in R and S should be the same (see first example).
NOTE 2: If $\forall i,j$, $R_i \neq S_j$, then $R * S |_P \downarrow R_1 \times \ldots \times R_n = R$ and $R * S |_P \downarrow S_1 \times \ldots \times S_m = S$.

Example: $A = \{1, 2\}$ $B = \{a,b,c\}$ $C = \{\alpha, \beta, \gamma\}$. $R = \{ (1,a), (2,c) \}$ $S = \{(a,\alpha), (b,\beta)\}$
$R * S |_{A \times B \times C} = \{ (1,a,\alpha) \}$
$R * S |_{A \times B \times B \times C} = \{(1,a,a,\alpha), (1,a,b,\beta), (2,c,a,\alpha), (2,c,b,\beta) \}$

DEFINITION 12 - Variable

Let N be a countable set with a generic element n (comprising some type of time measure), and $X \subseteq U$. A variable x of type X is a function $x : N \rightarrow X$. Note that a function is also a relation and hence it can be expressed by a set. Thus, $x \subset N \times X$.

Examples: $N = \{1, 2, 3\}$, $X = \{a, b, c\}$, $x(1) = a$, $x(2) = b$, $x(3) = c$ or $x = \{ (1, a), (2, b), (3, c) \}$

DEFINITION 13 - Composed Variable

Let x be a variable of type X. If the elements of X are n-tuples with $n > 1$, then x is called a composed variable (or structure).

The value of a composed variable, in a particular instant of time, will always be a tuple. The individual value of each sub-element of this tuple can be obtained by its reference index, the field of the variable. If $X = X_1 \times ... \times X_n$, then each field of x can be viewed as a free projection on $N \times X_i$, i.e., it is a standard variable of type X_i.

Example: $N=\{1, 2, 3\}$, $X_1 = \{a,b\}$, $X_2 = \{c,d\}$ $X = X_1 \times X_2 = \{ (a,c),(a,d),(b,c),(b,d)\}$

$x = \{ (1,(a,c)) , (2,(a,d)), (3, (a,d)) \}$

$x \downarrow N \times X_1 = \{ (1,a) , (2,a), (3, a) \}$

$x \downarrow N \times X_2 = \{ (1,c) , (2,d), (3, d) \}$

From the definition of variable we may proceed to define the concept of object. The mathematical concept of object is closely related to its intuitive physical meaning. Ontologically, an object is an entity of the real world and is characterized by its properties. The properties are its attributes (Wand, 1989). Based on a frame of reference, it is possible to find attributes distinguishing different objects. Thus attributes describe the objects. This view of objects does not consider that, in addition to its existence, the objects also “act” in real world. Therefore, a mathematical concept of object must model its active aspect.

The conceptualization of object cannot, in principle, be made in an independent way. Although we can imagine the existence of an object by itself, we should also consider its capability to interact with different objects. In other words, to introduce the main concepts about objects, we have to discuss object systems. An object system is a set of interacting entities. We assume both, existence and interaction, fulfilling the following principles:

- Objects are unique and are uniquely identified by their names,
- Each object has a set of attributes and/or parts,
- An object may have a set of transformation functions,
- An object may consume another object,
- An object may generate another object,
- Objects can be hierarchically classified,
- The interaction among objects are due to consumption and generation of objects.

The mechanisms behind object consumption and generation, and object internal parameter changes as well are as follows. Each object is assumed to have two types of interfaces: input and output interfaces. The input interface is composed by a collection of gates (input gates) which receives the objects to be consumed. These gates are selective, i.e., they select which objects are to be consumed at a time. Selection considers the simultaneous availability of all object needed to trigger a transformation function. When fired by the selection procedure, the transformation function is triggered. Triggering transformation functions may modify internal attributes of the object, and/or generate newer objects. The objects created are output by the output interface through output gates. An analogy may help to grasp the idea of this procedure. In object-oriented languages object consumption and generation procedures correspond to a method call. Each parameter passed to the method is an object to be consumed. The calling program should determine the value of each of these. The process of storing each parameter in a variable corresponds, in our case, to the creation of (passive) object-messages, one for each parameter, enabling the triggering of other object. The method function call corresponds to the selection function for the enabled object, and causes its triggering. The triggering mechanism call the transformation function of the object. It uses the internal variables of the triggered object, and the values given by the object-messages as function arguments. In the same way, it uses internal variables of the triggered object as output. Once the function is triggered, it can generate an output, which in our case corresponds to the generation of one or more object-messages They hold the messages as the values returned by the object method. Note some properties of the model proposed here. Object triggering is closely related with the selection function. With an appropriate implementation of selection functions, objects can become autonomous entities, i.e., independent of an external synchronization mechanism. Therefore, as soon as existing objects satisfy the conditions above, they are used. Nevertheless, one can implement different types of external synchronism using adequate selection functions. (which should be necessary sometimes, e.g., when two different objects want to consume the same object). Synchronization is needed when, for instance, two different objects are to consume the same object, or when an active object is consuming other object, and at the same time is being consumed. This would be an inconsistency. Another possibility is address based selection mechanisms: the object-message specifies which object will consume it. Actually, this is analogous of a method call in an object-oriented language. This selection mechanism can be used to create synchronization objects. In this strategy, no object is enabled until a synchronizer object (possibly created by a central controller object) exists. Synchronism, despite being useful sometimes, is not a requirement in object systems. The behavior of real objects, with asynchronous and parallel activities can be modeled. And more: note that both consumed and generated objects are not necessarily passive. This allows adaptive and self-organizing systems to be modeled by the object system model.

DEFINITION 14 - Class

A class C is a set whose elements c_i are tuples of the type:

$$(v_1, v_2, \ldots, v_n, f_1, f_2, \ldots, f_m), \ n \geq 0, m \geq 0$$

where $v_i \in V_i$, and f_j are functions

$$f_j : \bigtimes_{p \in P_j} V_p \rightarrow \bigtimes_{q \in Q_j} V_q .$$

Here $\bigtimes$ means the cartesian product, $P_j \subseteq \{1, \ldots, n\}$ and $Q_j \subseteq \{1, \ldots, n\}$.

DEFINITION 15 - Object

Let C be an non-empty class and c be a variable of type C. Thus c is an object of class C.

It is worth noting that an object, as a variable, supports objects composed by parts which are objects themselves. In addition, if n=1 and m=0 then the tuple reduces to a single element. Hence a standard variable (a primitive object) is an object. For an empty class n=m=0 and there is no object. Clearly structures are also object. As it will be seen later, a structure is a passive object.

DEFINITION 16 - Instance of an Object

Let c be an object of class C. The instance c(n) is the value of c at n.

C is a set of tuples. Therefore the instance of an object is an element of C, i.e. a tuple.

DEFINITION 17 - Superclass and Subclass

Let C be a class. The set D whose elements are sub-tuples of the elements of C belonging to the same universe, and each element in C is associated with one element of D and D is itself a class, is called a superclass of C. In this case C is a subclass of D.

Note that a class can be defined from primitive classes. Since class is a relation, another class can be generated by the cylindrical extension of a class, by the junction of two or more classes, or by both a junction and a cylindrical extension. In all cases, the primitive classes are superclasses of the newly generated class. Moreover, for a given a class its cylindrical extension is a subclass of itself. The junction of two classes is a subclass of both of them. Any class is a subclass of empty class. Therefore a hierarchy of classes is induced by projections, junctions and cylindric extensions, figure 5.

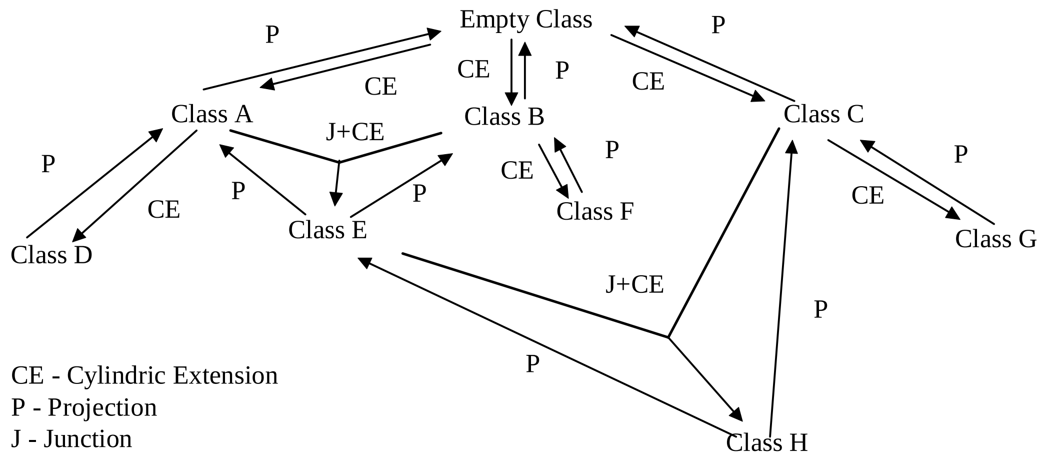

Figure 5 - Example of Class Hierarchy

DEFINITION 18 - Sub-object

Let c be an object of class C and d an object of class D a superclass of C. If for any n the instance of d is a sub-tuple of the instance of c, then d is a sub-object of c.

In other words, d is the free projection of c in $N \times D$, i.e. $d = c \downarrow N \times D$.

DEFINITION 19 - Active and Passive Objects

An object c of a class C is called an active object if $m > 0$. If $m = 0$, c is a passive object.

DEFINITION 20 - Input Interface

Let c be an active object of a class C and I a superclass of C, defined by:

$$I = \bigtimes_i V_i \ , \forall i \in \{1,\ldots,n\} \text{ such that } \begin{cases} \exists f_j ,1 \le j \le m \text{ where } i \in P_j \\ \forall f_l ,1 \le l \le m \ i \notin Q_l \end{cases}$$

The input interface i of the object c is the passive object generated by the free projection of c in $N \times I$, i.e. $i = c \downarrow N \times I$.

DEFINITION 21 - Function Specific Input Interfaces

Let c be an active object of class C, i its input interface, and I^j a superclass of I and C such that :

$$I^j = \bigtimes_i V_i \ , \forall i \in \{1,\ldots,n\} \text{ such that for } f_j \quad i \in P_j \text{ e } \forall l \in \{1,\ldots,m\},\ i \notin Q_l$$

The specific input interface for function j of c, i^j, is the free projection of c in $N \times I^j$.

Note that $i^j = c \downarrow N \times I^j = i \downarrow N \times I^j$. If the elements of class C have m functions, then there exist m function specific input interfaces. Each i^j is a sub-object of i and c.

DEFINITION 22 - Output Interface

Let c be an active object of class C and O a superclass of C characterized by:

$$O = \bigtimes_i V_i \ , \forall i \in \{1,\ldots,n\} \text{ such that } \begin{cases} \exists f_j ,1 \le j \le m \text{ where } i \in Q_j \\ \forall f_l ,1 \le l \le m \ i \notin P_l \end{cases}$$

The output interface o of object c is the passive object generated by the free projection of c in $N \times O$, i.e. $i = c \downarrow N \times O$.

DEFINITION 23 -Function Specific Output Interfaces

Let c be an active object of a class C, o its output interface, and O^j is a superclass of O and C such that :

$$O^j = \bigtimes_i V_i \ , \forall i \in \{1,\dots,n\} \text{ such that for } f_j \quad i \in Q_j \text{ e } \forall l \in \{1,\dots,m\},\ i \notin P_l$$

The output interface specific for function j of c, o^j , is the free projection of c in $N \times O^j$.

Clearly, $o^j = c \downarrow N \times O^j = o \downarrow N \times O^j$ and if the elements of class C have m functions, then there exist m function specific input interfaces. Each o^j is a sub-object of o and c.

DEFINITION 24 - Existence of Objects

An object c is said to exist at n if the function which maps instances of c in C is defined for $n \in N$.

DEFINITION 25 - Generation and Consumption of Objects

An object is generated at n if it does not exist at n and does exist at n+1. An object is consumed at n if it do exist at n and does not exist at n+1.

DEFINITION 26 - Enabling Scope of Functions

Consider an active object c from a class $C = \{ (v_1, v_2 , \dots , v_n , f_1, f_2 , \dots , f_m)\}$, a function f_j of C, and i^j an input interface specific for function f_j . Let β be the arity of instances of i^j , gi an input index function for f_j , $gi : \{1, \dots , \beta \} \rightarrow \{1, \dots , n\}$ mapping each component from instances of the input interface specific for f_j to a component in instances of c, and $B = \{0,1\}$. An enabling scope for f_j is a set of tuples $H = \{(h_t ,b_t)\}$, $t = 1, \dots , \beta$, where h_t is an object of class $V_{gi(t)}$ and $b_t \in B$ is a boolean value indicating if object h_t should ($b_t = 1$) or should not ($b_t = 0$) be consumed when c triggers.

DEFINITION 27 - Generative Scope of Functions

Assume an active object c of class $C = \{ (v_1, v_2 , \dots , v_n , f_1, f_2 , \dots , f_m)\}$, a function f_j of C, an output interface o^j specific or function f_j , α the arity of instances of o^j , and an output index function for f_j, $go : \{1, \dots , \alpha \} \rightarrow \{1, \dots , n\}$ mapping each component from instances of output interface specific for f_j to a component in instances of c. A generative scope for f_j is a set of objects $S = \{s_u \}$, $u = 1, \dots , \alpha$, where s_u is an object of class $V_{go(u)}$.

DEFINITION 28 - Enabling of an Active Object

An active object of class C is enabled at n if all objects belonging to an enabling scope of one of its functions f_j do exist at n. Function f_j is said to be enabled at n.

DEFINITION 29 - Triggering of an Active Object

Consider the following:

a)-an object c from a class C,

b)- the instance of c at n, $c(n) = (v_1 (n), \dots , v_n (n), f_1 (n), \dots , f_m (n))$,

c)-a function f_j of c at n, enabled by $H = \{(h_t ,b_t)\}$,

d)-a generative scope $S = \{s_u \}$ for f_j such that if $s \in S$, then or s does not exist at n, or $s \in H$,

e)-p, the number of values such that $k \in P_j$, $k = 1, \dots , n$, (??????)

f)-a domain index function $gd : (1, \dots , p \} \rightarrow \{1 , \dots , n\}$ for f_j ,

g)-the projection of f(.) on V_k , $f(.)\downarrow V_k$,

h)-β, α, gi e go as before.

Triggering an active object at n means:

1)- determination of c's instance at instant n+1, given the instance of c at n and the instances of h_t at n:

$$v_i(n+1) = \begin{cases} v_i(n) \text{ if } i \notin P_j \text{ and } i \notin Q_j \\ h_{gi^{-1}(i)}(n) \text{ if } i \in P_j \text{ and } i \notin Q_j \\ f_j(w_1,\dots,w_b) \downarrow V_i \text{ if } i \in Q_j \end{cases}$$

$$\text{where } w_r = \begin{cases} h_{gi^{-1}(gd(r))}(n) \text{ , if } gd(r) \in P_j \\ v_{gd(r)}(n) \text{ , if } gd(r) \notin P_j \end{cases}$$

2)- consumption ($b_t = 1$) of object h_t , $(h_t , b_t) \in H$,at n,

3)-generation, at n , of the objects of S that do not exist at n,

4)-determination of the instances of the objects of S, at n+1, according to the instances of c at n+1:

$$s_u(n+1) = v_{go(u)}(n+1)$$

DEFINITION 30 - Object System

Assume the following:

a)-c_i are objects of class C_i , i = 1, ... , δ ,

b)-$C = \bigcup_i c_i$,

c)-$\Theta_i = \{ 0, \dots , m_i \}$ where m_i is the number of functions for object c_i,

d)-B = {0,1},

e)-γ_i , $0 \le i \le \delta$, $\delta > 0$, are selection functions $\gamma_i : N \to 2^{c \times B} \times 2^c \times \Theta_i$ which, for each object c_i at n, select an enabling scope H_i , a generative scope S_i and a function index such that $\forall(c,b) \in H_i$, if b = 0, then $(\forall k \neq i)((c,1) \notin H_k)$ and if b = 1, then $(\forall k \neq i)$ $((c,0) \notin H_k$ and $(c,1) \notin H_k)$, $\forall c \in S_i$, $(\forall k \neq i)(c \notin S_k)$ and $(\forall k)((c,1) \notin H_k)$. H_i is an enabling scope and S_i is a generative scope for function $\gamma_i(n) \downarrow \Theta_i$. If c_i is a passive object or, at n $\nexists H_i \neq \emptyset$ or $\nexists S_i \neq \emptyset$, or if $(\exists k \neq i)$ then $((c_i , 1) \in H_k)$, then $\gamma_i(n) = (\emptyset, \emptyset, 0)$. The third index is null indicating that no function is to be executed. The meaning of these conditions are grasped as follows. An object of an enabling scope can not be consumed by any other object at the same instant. If an object is to be consumed, then it can not be in any other enabling scope. If an object is part of a generative scope of an active object, then it cannot be in any other generative scope. If the object in question is passive, or if active it does not have an enabling scope for any of its functions, or it is to be consumed in the current instant, then its selection function must do nothing.

An object system Ω is a set of pairs $\{\omega_i\}$, $\omega_i = (c_i ,\gamma_i)$, such that :

1)-for n=0, there exists at least one ω_i with an object c_i defined,

2)-for n>0 all active objects c_i with $\gamma_i(n) \neq (\emptyset, \emptyset, 0)$, i.e. objects whose selection functions are in the form $\gamma_i(n) = (H_i, S_i, j)$ may trigger according to its enabling scope H_i and the generative scope S_i , using its j-th internal function,

3)-for n>0, all objects c_i which exist at n and do not have its instance at (n+1) determined by item 2, may be regenerated with $c_i(n+1) = c_i(n)$.

The definition of an object system is actually a specification. To call a given set of objects an object system, the values of its instances (and their existence at different times) should be associated with each other according to the laws of triggering and regeneration of objects given by itens 2 and 3 of definition. The laws determine, at each time instant, the values of the instances (or its inexistence) based on their previous values. The objects that interact at each instant are determined according to the selection functions, which define the enabling scopes, the generative scopes, and the functions to be triggered for each active object. This interdependence among objects is illustrated in figure 6.

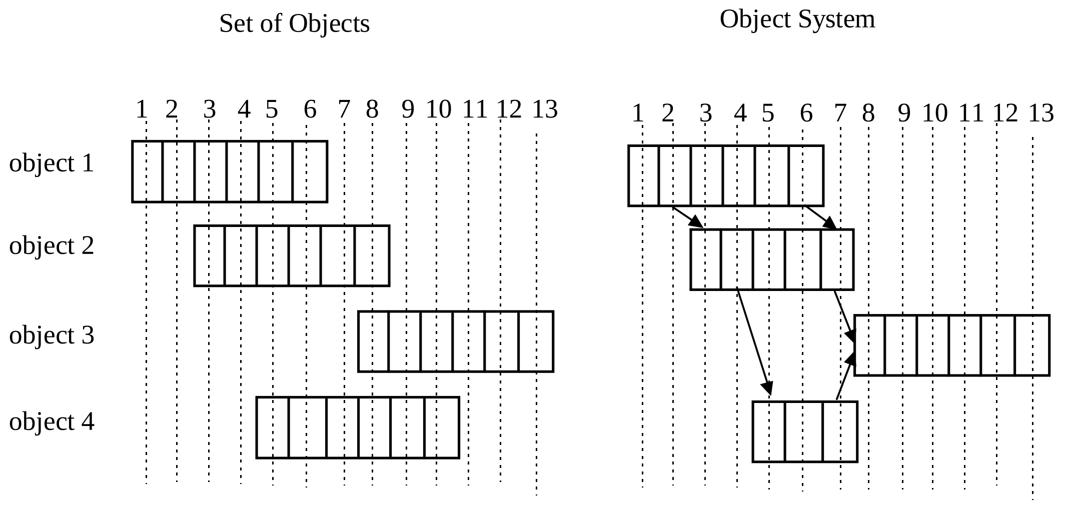

Figure 6 - Example of an Object System

In figure 6 each object is represented by a sequence of rectangles, each rectangle corresponding to a temporal instance of an object. When an object is not defined at an instant it is not shown. For elements within the set of objects, the values of the instances keep no relation among them. In contrast, in the object system the values of the instances at successive instants are related because they depend on the previous values of objects instances. Triggering is represented by arrows. Thus, from instant 1 to 2 there is no triggering, only the regeneration of the instance for object 1. From instant 2 to 3 an object (not shown in the figure) is triggered using object 1 as its enabling scope and object 2 as its generative scope. The result is the generation of object 2 and the definition of its instance value for instant 3. In this case object 1 was not consumed, but regenerated for instant 3. From instant 3 to 4 only regeneration occurs. At instant 5, objects 1 and 2 are regenerated and object 4 is generated, object 2 being used as an enabling scope. From instant 5 to 6, objects 1, 2 and 4 are regenerated. From instants 6 to 7 the object 1 is consumed, and determines the instance of object 2 (by the triggering of an object not shown in figure, which has object 1 as its enabling scope, and object 2 as its generative scope). From instants 7 to 8, objects 2 and 4 are consumed, and object 3 is generated. These modifications are due to some object (not shown) triggering ,with objects 2 and 4 in its enabling scope, and object 3 as its generative scope. From instant 8 to 13 (final instant) only regeneration of object 3 occurs.

The determination, at each instant, of which objects should participate in each enabling and/or generative scope is up to the selection function. Therefore, the selection function plays a fundamental role in the object system dynamics.

Objects are functions and thereare, in principle, no restrictions on the their form. For a set of objects the functions can be any. An object system has, as opposed to a set of objects, a clear recursive nature. The recursiveness is not simple because objects may be *partial* functions, i.e. they do not have to be defined for any value of its domain. This rises the fundamental question about the computability (Davis, 1958; Epstein & Carnielli, 1989) of object systems, certainly a desirable property. An object system being computable means that we can determine, for each $n \in N$, the values of the instances for the existing objects at n. Conditions for computability of object systems are given below.

THEOREM 1 - Computability of Object Systems

Let Ω be an object system. If Ω has a finite number of elements ω_I , and all selection functions γ_I are computable, and all internal functions of all objects c_i of ω_i are computable, then Ω is computable.

PROOF: The proof is by induction. Suppose that the number of objects for an object system is P. Consider the function $c_i'(n)$ such that:

$$c_i'(n) = \begin{cases} c_i(n), \text{ if } c_i \text{ is defined in n} \\ \varepsilon, \text{ otherwise} \end{cases}$$

We may regard c_i' (n) as an extended object. The value of its instance at n is:

$$c_i'(n) = f(c_1'(n-1), c_2'(n-1), \ldots , c_P'(n-1)).$$

The function f is determined either by the triggering of some object at n-1 or by regeneration. If γ_i is computable and P is finite, then it is possible to find, for all active objects of Ω at n, the enabling scope H_i , the generative scope S_i , and the internal function ϕ_i to be triggered. If all internal functions are computable, then ϕ_i is also computable. If (c_i , 1) appears at the enabling scope H_j of object c_j , then c_i must be consumed, which means that it should not exist at n. Thus, f(.) = ε. If c_i appears in a generative scope S_j of object c_j, then f(.) = $\phi_j(c_{k_1}', c_{k_2}', \ldots, c_{k_m}')$, where each c_{k_i}' belongs to H_j .From the definition of selection function, each object c_{k_i} belonging to the enabling scope should exist at triggering. Hence c_{k_i}' = c_{k_i}, which means that $\phi_j(c_{k_1}', c_{k_2}', \ldots, c_{k_m}')$ = $\phi_j(c_{k_1}, c_{k_2}, \ldots, c_{k_m})$. If c_i does not appear in any scope (enabling or generative) of any active object, then f(.) = c_i (n-1) (regeneration law). Therefore, we conclude that the function f is computable.

Given an initial instance for each extended object $c_i'(0)$, we can compute any instance $c_i'(n)$ for the object. In other words, the extended objects c_i' are computable. If we associate each extended object c_i' with an object c_i , then, the objects c_i are computable. Since c_i (n) and γ_i (n) can both be computed for all n if c_i does exist in n, ω_i can determined meaning that Ω is computable.

The conditions stated in theorem 1 to guarantee object system computability are sufficient conditions only. To be computable, an object system do not need to necessarily have a finite number of objects. If for adjacent instants n and n+1 the number of existing objects is finite, then the system is computable. The proof for this statement is analogous to the one above if we remove the objects that do not exist at n and n+1 because they can not affect or be affected by triggering at n.

An object system fulfilling the conditions of theorem 1 can be viewed as the limit of a (possibly infinite) sequence of objects systems Ω_1 , Ω_2 , ... , each Ω_i with a finite number of objects defined on a finite and incremental domain N_i. That is:

$N_0 = \{0\}$

$N_i = N_{i-1} \cup \{i\}$.....

Consequently, each object system Ω_i is computable and the whole sequence is computable as well. The infinite object is the i-th element of this sequence when $i \rightarrow \infty$. Hence, by induction, the object system is computable although infinite.

An object network is a special type of object system in which additional restrictions concerning interactions are included. To distinguish object network and object system let us assume places and arcs whose role is similar to those in Petri nets. Objects in places can only interact with objects in places connected through arcs. Thus, at each instant, the objects defined should be at one place. For each place there is a set of places connected to it by means of input arcs. These places are called the input gate of the place. Objects that are in this place can only have in its enabling scope objects that are in its input gates. More specifically, for each field of input interface of objects in this place there should exist one corresponding input gate. In the same way, for each place, there is a set of places that are connected to the former by means of output arcs, which are called output gates. For each field of output interface of objects in this place there should exist one corresponding output gate. With those conditions we can see that, for each place, there should be only objects of the same class. Besides those special characteristics, an object network is similar to an object system.

DEFINITION 31 - Object Network

Assume the following:

a)- a set of classes $\Sigma = \{C_i\}$

b)- a set of objects $\mathcal{C} = \{c_i\}$, where c_i are objects from a class C_i , $C_i \in \Sigma$, $0 \le i \le \delta$, $\delta > 0$.

c)- $\Pi = \{ \pi_i \}$ a set of places π_i

d)- A , a set of arcs $A = \{a_i\}$

e)-η a node function $\eta : A \rightarrow \Pi \times \Pi$

f)-ξ a localization function $\xi : N \times \mathcal{C} \rightarrow \Pi$, which relates to each object $c \in \mathcal{C}$, for each instant n, a place π.

g)-$F(\pi)$ a mapping $\Pi \rightarrow 2^{\Pi}$, defined by $F(\pi) = \cup \pi_k$ where $k \in K$, $K = \{k \mid \exists a_j \in A$ such that $\eta(a_j) = (\pi_k,\pi) \}$.

h)-$V(\pi)$ a mapping $\Pi \rightarrow 2^{\Pi}$, defined by $V(\pi) = \cup \pi_k$ where $k \in K$, $K = \{k \mid \exists a_j \in A$ such that $\eta(a_j) = (\pi,\pi_k) \}$.

i)- $X(\pi)$ a mapping of connections $\Pi \rightarrow 2^{\Pi}$, such that $X(\pi) = F(\pi) \cup V(\pi)$.

j)-$\Xi = \Xi(\pi)$ a mapping of classes $\Pi \rightarrow \Sigma$, such that $\forall\pi \in \Pi$ for each field v_i of input interface of objects from class $\Xi(\pi)$, being v_i an object from class C, $\exists\pi_k$, $\pi_k \in F(\pi)$, such that $\Xi(\pi_k) = C$, and for each field v_i of output interface of objects from class $\Xi(\pi)$, being v_i an object from class C, $\exists\pi_k$, $\pi_k \in V(\pi)$, such that $\Xi(\pi_k) = C$.

k)- i^i the input interface of an object from class $\Xi(\pi_i)$.

l)-o^i the output interface of an object from class $\Xi(\pi_i)$.

m)-∂_i the number of fields in i^i and ρ_i the number of fields in o^i .

n)-the function fpi_i the attribute function for input gates of objects which are at a place π_i , defined $fpi_i : \{1, \dots , \partial_i \} \rightarrow A$ and $fpi = \{fpi_i \}$.

o)-$fpo_i = \{ 1, \dots , \rho_i \} \rightarrow A$ the attribute function for output gates of objects that are at a place π_i and $fpo = \{fpo_i \}$

p)-$\Theta_i = \{ 0, \dots , m_i \}$, where m_i is the number of function for object c_i

q)-$\gamma = \{ \gamma_i \}$, $0 \le i \le \delta$, $\delta > 0$, which elements are selection functions $\gamma_i : N \rightarrow 2^{c \times B} \times 2^{c} \times \Theta_i$ that for each object c_i , in an instant n, select an enabling scope H_i , a generative scope S_i and the index for the function to be executed by the object, having as a restriction that $\forall(c,b) \in H_i$, $\xi(n,c) = \pi$, $\pi \in F(\xi(n,c_i))$, if $b = 0$, $(\forall k \neq i)((c,1) \notin H_k)$ and if $b = 1$, $(\forall k \neq i)$ $((c,0) \notin H_k$ e $(c,1) \notin H_k)$, $\forall c \in S_i$, $\xi(n,c) = \pi$, $\pi \in V(\xi(n,c_i))$, $(\forall k \neq i)(c \notin S_k)$ and $(\forall k)((c,1) \notin H_k)$. More than that, H_i should be an enabling scope and S_i should be a generative scope for function f_k , $k = \gamma_i(n) \downarrow \Theta_i$. If c_i is a passive object or, for a given n, $\nexists H_i \neq \emptyset$ or $\nexists S_i \neq \emptyset$, or if $(\exists k \neq i)((c_i , 1) \in H_k)$ then $\gamma_i(n) = (\emptyset, \emptyset, 0)$. The third index being 0 does mean that no function is going to be executed. Those conditions are analogous to the selection function for an object system, added by the following conditions: Any object belonging to the enabling scope of another object should be connected to the place where the active object is by an input arc. Any object belonging to the generative scope of an active object should be put in a place that is connected to the place where the active object is by means of an output arc.

An object network $\Re$ is a tuple $\Re = (\Sigma, \Pi, \Xi, A, \eta, fpi, fpo, \mathcal{C} , \xi, \gamma)$, such that:

1)- an objects system $\Omega = \{ (c_i , \gamma_i) \}$ is determined by choosing $c_i \in \mathcal{C}$ and $\gamma_i \in \gamma$, $0 \le i \le \delta$,

2)-for each object $c_i \in \mathcal{C}$ with a function f_j being triggered at n, being this object at n at a place $\pi = \xi(n,c_i)$, the objects s_i^k belonging to the generative scope S_i indicated by $\gamma_i(n)$ should have a localization function defined by:

$$\xi(n+1,s_i^k) = \pi^k$$

where π^k should be such that $\eta(fpo_{\pi}(k')) = (\pi,\pi^k)$ and k' is the index of the k-th field of the input interface specific to function f_i of c_i referred at the output interface of c_i .

Alike an object system, an object network can be seen as an specification comprising the trajectories over time for a set of objects. In the same way, an important class of object networks are those that are computable. Again, we can determine a computable object network iteratively, generating a sequence of object networks $\Re_0$, $\Re_1$, ... , where each $\Re_i$ contain a finite number of objects, defined over incremental domains N_i . Each object network from the sequence is equivalent to the previous, being its objects added by a temporal instance, according to the laws of triggering, the selection function defined for the last instance and the localization function. The procedure is similar to the one used for objects systems with infinite number of objects, included the conditions for localization functions. With this methodology, one can determine, for each instant $n \in N$, the instance of any object that exists in n. The initial objects network from the sequence, $\Re_0$ has a special denomination, being called the kernel of an object network. In its kernel, objects and localization functions are defined only for instance n=0, and the selection function should be computable, being described by an algorithm γ'.

DEFINITION 32 - Kernel of an Object Network

We define the kernel of an objects network as an objects network $\Re_0 = (\Sigma, \Pi, \Xi, A, \eta$, fpi, fpo, $\mathcal{C}^0$, ξ^0 , $\gamma)$, where

- $\Sigma, \Pi, \Xi, A, \eta$, fpi and fpo are as in the definition of objects network and
- $\mathcal{C}^0 = \{c_i'\}$ is a set of objects defined only for n=0.
- ξ^0 is a localization function $\xi^0 : N \times \mathcal{C}^0 \rightarrow \Pi$, defined only for n=0.
- γ is a selection function determined by means an algorithm γ'.

Starting with a kernel, new finite object networks are calculated in sequence. Each algorithm step generates a new network that is, fundamentally, the previous one increased by the definition of new values for $\mathcal{C}$, ξ and γ . The new $\mathcal{C}$ is the old one plus the objects to be generated at step n, according to γ. The new ξ is also the old one, defined now also for instant n, again according to γ. The algorithm γ' defines incrementally function γ. So, at each step the algorithm defines new instances for the current objects and new values for localization function and selection function such that for each instant n, we obtain new $\mathcal{C}$, ξ and γ corresponding to a new finite objects network. Finite networks correspond to some element of such a sequence. Infinite networks (both networks with a finite number of objects, defined in infinite domains, and networks with an infinite number of objects), are considered as the limit element i of the sequence, when $i \rightarrow \infty$. An example of such algorithm is described in figure 7:

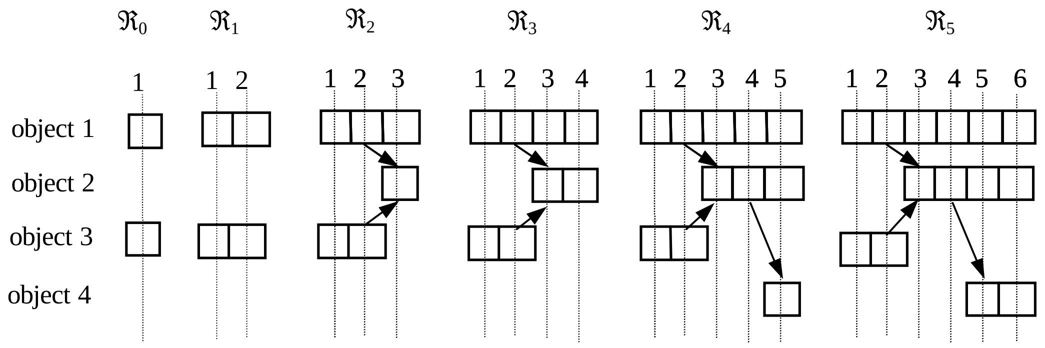

Figure 7 - Example of an Evolution Sequence

Examples of main algorithms described through pseudo-codes are the following:

```
procedure Main:
{Define $\mathcal{C}$ , composed by objects $c_i$ given in $\mathcal{C}^0$ .
 Define localization function $\xi$, composed by the tuples in $\xi^0$ .
 Define $\gamma = \emptyset$
 For n from 0 to final
   {Apply $\gamma'$ to determine $\gamma(n)$ and refresh $\gamma$.
    For all active objects $c_i$ existing at n:
      {Calculate $\gamma_i(n) = (H_i , S_i , f)$.
       If $H_i = \emptyset$, go to next object
       If $H_i \neq \emptyset$ :
         {execute function f, generating a new instance $c_i(n+1)$
          refresh $c_i$ : $c_i = c_i(n) \cup c_i(n+1)$.
          For all $s_i^k \in S_i$
            {If $s_i^k \notin \mathcal{C}$ generate a new empty object and add to $\mathcal{C}$
             calculate the value for $s_i^k(n+1)$ from $c_i(n+1)$ and refresh $s_i^k$ .
             determine $\xi(n+1,s_i^k) \in V(\xi(n,c_i))$ and refresh $\xi$.
            }
         }
      }
    For all objects $c_i$ such that ($c_i$ , 1) does not appear at any other enabling scope and $c_i$ is not at
    neither generative scope
        $c_i$ (n+1) = $c_i$ (n).
    }
}


Procedure $\gamma$'
{For each active object $c_i$
    {For each function $f_j$ of active object $c_i$
        {Generate a new empty enabling scope for function $f_j$
          For each field k of input interface specific to $f_j$
            {check if in the place pointed by the arc fpo(k) does exist:
                no object, one object or  more than one object
              If there is no object, delete the enabling scope(s) and go to next function
              If there is only one object, put it into the enabling scope(s).
              If there is more than one object, do as many copies of the enabling scope as the number of
                objects and put one object into each enabling scope.
            }
          For each enabling scope calculate a performance index
        }
    }
  For each active object $c_i$
    {Do a list ordered by the performance index containing the function and the enabling scope.
    }
  For each active object $c_i$
    {Choose the first element in the list as the function and enabling scope for the object
     Check if is there any conflict with other objects choices.
     If is there a conflict, use a decision criteria. The looser object chooses than the next in its list.
     If the own object $c_i$ belongs to the enabling scope of another object, cancel its enabling scope and
       reorganize the list.
    }
For each active object $c_i$ with an enabling scope different from empty
    {Create an empty generative scope for $c_i$
     For each field k of output interface specific to function $f_j$ chosen
        {If is there an object in $C$ not defined at n-1 and n for the desired class,
            put it into the generative scope, else create a new object for the desired class and include it.
        }
    }
 Returns, for each object, the enabling scope, the generative scope and the function chosen
}
```

7. Conclusions

In this paper, we presented the foundations of computational semiotics. Based on semiotics, a traditional human science discipline, we derived a hierarchy of knowledge types. This hierarchy arised from two main classes. The first results from the elementary classification of knowledge to distinguish the different kinds according to its significant aspects. The second, classifies knowledge as it is used in in intelligent systems (i.e. applied knowledge). Both classes were described informally here. Next, we proposed a family of notions whose aim was to be put forward a foundation for a theory of objects. Among the notions we emphasized the concepts of object, object system and object network, which were appropriately formalized by convenient mathematical structures. The purpose of the formalization was two fold. First, we formalized the concept of object. Second, object network was set as an alternative way to represent the dynamic characteristics of systems that change their own structure. This avoids limitations of traditional modeling tools, like finite state automata and Petri nets, etc.. In this sense, the proposed model allows a very natural representation for this class of systems.

The hierarchy of types of knowledge and the mathematical theory of objects are the foundations for computational semiotics. In part II we use the concepts of the mathematical theory of objects to formalize the different types of knowledge given in the hierarchy derived here. In addition, part II offers an application example, in which knowledge from different types are used to model an intelligent system to control an autonomous vehicle in an unknown environment. The purpose is to illustrate the use of computational semiotics as a new approach to build intelligent systems.

8. References

(Albus, J.S. 1991) - “Outline for a Theory of Intelligence” - IEEE Transactions on Systems, Man and Cybernetics, vol. 21, n. 3, May/June 1991.

(Anderson, J.R. 1989) - "A Theory of the Origins of Human Knowledge" - Artificial Intelligence 40 (1989) pp. 313-351.

(Barr, A. ; Cohen, P. ; Feigenbaum, E.A. 1981) - “The Handbook of Artificial Inteligence”, vol. IV, Addison-Wesley Publishing Company, 1981, 2nd edition 1989.

(Barr, A. ; Feigenbaum, E.A. 1981a) - “The Handbook of Artificial Inteligence”, vol. I, Addison-Wesley Publishing Company, 1981, 2nd edition 1989.

(Barr, A. ; Feigenbaum, E.A. 1981b) - “The Handbook of Artificial Inteligence”, vol. II, Addison-Wesley Publishing Company, 1981, 2nd edition 1989.

(Boden, M. A. 1983) - “As Idéias de Piaget” - tradução de Álvaro Cabral - Editora Cultrix - Editora da Universidade de São Paulo - São Paulo, 1983.

(Booker, L.B. ; Goldberg, D.E. ; Holland, J.H. 1989) - “Classifier Systems and Genetic Algorithms” - Artificial Intelligence 40 (1989) pp. 235-282.

(Brachman, R.J. ; Schmolze, J.G. 1985) - "An Overview of the KL-ONE Knowledge Representation System" - Cognitive Science 9, 171-216 (1985).

(Brooks, R.A. 1991) - “Intelligence Without Reason” - Proceedings of the Twelfth International Conference on Artificial Intelligence, Vol. 1 (1991) 569-595.

(Chang, C.L. ; Lee, R.C.T. 1973) - “Symbolic Logic and Mechanical Theorem Proving” - Academic Press, New York, 1973.

(Coelho Netto, J.T. 1980) - “Semiótica, Informação e Comunicação” - Coleção Debates - tomo 168 - Editora Perspectiva.

(Cohen, P. ; Feigenbaum, E.A. 1981) - “The Handbook of Artificial Inteligence”, vol. III, Addison-Wesley Publishing Company, 1981, 2nd edition 1989.

(Collins, A. ; Michalski, R. 1989) - "The Logic of Plausible Reasoning : A Core Theory" - Cognitive Science 13, 1-49 (1989).

(Davis, M. 1958) - “Computability & Unsolvability” - McGraw-Hill Book Company, New York, 1958.

(Eco, U. 1980) - “Tratado Geral de Semiótica” - Coleção Estudos - tomo 73 - Editora Perspectiva.

(Enderton, H. 1972) - “A Mathematical Introduction to Logic” - Academic Press, 1972, USA.

(Epstein, R.L.; Carnielli, W.A. 1989) - “Computability : Computable Functions, Logic and the Foundations of Mathematics” - Wadsworth & Brooks/ Cole Advanced Books & Software - Pacific Grove, California, USA.

(Goldberg, D.E. 1989) - “Genetic Algorithms in Search, Optimization, and Machine Learning” - Addison-Wesley Publishing Company, Inc.

(Gudwin, R.R. 1996) - “Contributions to the Mathematical Study of Intelligence” - Ph.D. Thesis, Departamento de Computação e Automação Industrial, Faculdade de Engenharia Elétrica, Universidade Estadual de Campinas, Campinas - SP, Brasil. (in portuguese).

(Kosko, B. 1992) - “Neural Networks and Fuzzy Systems - A Dynamical Systems Approach to Machine Intelligence” - Prentice-Hall International, Inc., London, UK.

(Laird, J.E. ; Newell, A. ; Rosenbloom, P.S. 1987) - “SOAR: An Architecture for General Intelligence” - Artificial Intelligence 33 (1987) 1-64.

(Meystel, A.M. 1996) - “Intelligent System as an Object of Semiotic Research” - 1996 Biennial Conference of the North American Fuzzy Information Processing Society - NAFIPS - New Frontiers in Fuzzy Logic and Soft Computing - Berkeley, California, USA, June 1996.

(Morris, C.W. 1947) - “Signs, Language and Behavior” - New York : Prentice Hall, 1947

(Morris, C. W. 1964) - “Significant and Significance”

(Morris, C. W. 1971) - “Foundation for a Theory of Signs” - in “Writings on the General Theory of Signs” - The Hague : Mouton, 1971

(Newell, A. 1982) - “The Knowledge Level” - Artificial Intelligence 18 (1982) 87-127.

(Nilsson, N. 1980) - “Principles of Artificial Intelligence” - Tioga, Palo Alto, CA, 1980.

(Norman, D.A. 1991) - “Approaches to the Study of Intelligence” - Artificial Intelligence 47 (1991) 327-346.

(Pedrycz, W. 1989) - “Fuzzy Control and Fuzzy Systems” - John Wiley & Sons Inc. , New York, USA.

(Peirce, 1960) - “Collected Papers of Charles Sanders Peirce” - vol I - Principles of Philosophy; vol II - Elements of Logic; vol III - Exact Logic; vol IV - The Simplest Mathematics; vol V - Pragmatism and Pragmaticism; vol. VI - Scientific Metaphysics - edited by Charles Hartshorne and Paul Weiss - Belknap Press of Harvard University Press - Cambridge, Massachussets, 2nd printing, 1960.

(Plato, 1991) - Platão - Diálogos - Coleção “Os Pensadores” - Seleção de textos de José Américo Motta Pessanha; tradução e notas de José Cavalcante de Souza, Jorge Paleikat e João Cruz Costa - 5. ed. São Paulo : Editora Nova Cultural - 1991.

(Turing, A.M. 1950) - “Computing Machinery and Intelligence” - Mind, 59:433-460, 1950, reproduzido em Collins, A. & Smith, E.E. - Readings in Cognitive Science - A Perspective from Psychology and Artificial Intelligence - Morgan Kaufmann Publishers, Inc. San Mateo, California, 1988

(Wand, Y. 1989) - “A Proposal for a Formal Model of Objects” - in Object-Oriented Concepts, Databases and Applications, W. Kim and F. Lochovsky, eds., ACM Press, New York, 1989, pp. 537-559

(Yager, R. Zadeh, L. 1992) – “An Introduction to Fuzzy Logic Applications in Intelligent Systems” – Kluwer Academic Publishers, Netherlands, 1992.

(Zurada, J.; Marks II, R.J.; Robinson, C.J. 1994) - “Computational Intelligence - Imitating Life” - IEEE Press, USA, 1994.