# Emergent inference, or how can a program become a self-programming AGI system?

Sergio Pissanetzky

Graduate School. Dep. of Physics. Texas A&M University
College Station, Texas 77843, USA. Retired
sergio@SciControls.com http://www.scicontrols.com

**Abstract.** A singer sings a song. But a record player can not sing. The record it plays captures only the *output* from the singer's ability to sing, not the ability itself. Likewise, a program does not capture its programmer's ability to program. If an AGI system is expected to possess the ability to write that program, then that program can not be the AGI system. If the AGI system is not expected to possess that ability, then it would not be an AGI system. Proposed in this paper is a self-programming AGI system where programs result as a *side effect* of an unrelated dynamical process. The system is supported by a new mathematical theory, where a *partially ordered set* is used as a universal knowledge base, and a *functional* is defined over the knowledge base. When the dynamical process minimizes the functional, the knowledge becomes organized and structures emerge. The structures are, in turn, a partially ordered set, and have their own functional. Iteration results in a nested *inheritance hierarchy* like the ones used in object-oriented programs. This hierarchy is the resulting program. There is evidence suggesting that the brain works in the same way. The dynamics is easy to simulate on any computer.

**Keywords:** AGI, self-programming, emergence, brain, intelligence

## 1 Introduction

This introductory Section examines several major technologies, some critical for the future of computational autonomy, that have been slow to develop and less than successful, and argues that they require no less than a full automation of the human analyst [1], and, as such, are positioned at the very heart of AGI.

**Self-programming.** In spite of enormous gains in computer power and mathematical methods, computers still depend on humans as they always have. Examples are the robot that serves wine from a bottle but the instruction that serves the wine precedes the one that removes the cork, and the Mars Rover that ignores the dancing martian [2]. Deep Blue plays chess better than any of its creators. Watson plays a quiz game better than any of its creators. But can they play checkers, or even learn how to? Their creators can. Self-programming programs do not fare any better. Ikon Flux [2] defines a world of which it itself

is a part and tries to learn from it and control it by constantly reprogramming itself. It can control the stage of a theatrical performance. But can it learn, without human intervention, how to control the Mars Rover? Or how to play tic-tac-toe? Its creators can. Its authors report difficulties with full automation, particularly with the two central issues we all have difficulty with: *refactoring* and *integration*.

**Refactoring is a universal phenomenon.** *Refactoring* [3] is a bottom-up procedure where an analyst finds *analogies* in data and uses them to *associate* and *organize* the data and to create *structures* or *objects* that are functional and understandable. The term was initially reserved for object-oriented (OO) software, but was later extended to non-OO software [4] and non-software systems such as the law [5], and even bacteria [6]. Refactoring was recognized as a universal phenomenon. We all practice refactoring all the time for tasks such as *planning*, where certain information is given and a plan needs to be prepared. However, refactoring has not been automated, not even for OO software. Tools are available to help with routine tasks, but only a human analyst knows how to find the analogies and organize the information into structures. The ability to refactor and form objects has not been captured in programs and remains privy to the human brain.

**Systems integration is also a universal phenomenon.** Closely related to refactoring is *Systems Integration*, a top-down procedure for combining two systems, usually man-made and equipped with interfaces. The systems are linked by operationally interconnecting their interfaces, and are expected to behave as one. Integration is a general engineering practice, routinely used for all kinds of systems. But integrating the interfaces is not easy. Integration is practiced by human integration engineers. It has so far resisted full automation, and is currently the subject of intense investigation, just as refactoring used to be about 20 years ago. Tools and methods have been developed, particularly in the field of narrow AI, but they require human supervision.

**Formal mathematics.** Formal mathematical methods such as mathematical logic and inductive inference, are very powerful theories, indeed. But in AGI, one should be more concerned about the abilities of the scientists that created the theories, or rather, the abilities of their brains. If it were possible to capture the abilities, then the theories would follow. But that has not happened yet. The abilities are not in the theories, and as a result, the theories have not been as productive as they were expected to be.

**Associative memories, ontologies, and the semantic web.** These problems are currently of enormous practical interest, and they all rely on one core issue: the ability to find analogies in knowledge. Once the analogies are found, they can be used to create associations and to organize the knowledge and form

structures. But analogy-making remains a unique ability of the human brain, not yet captured in any tool. As a result, associative memories, ontologies, and the semantic web are all being developed manually, with help from the tools.

**The common root.** Just as the singer's ability to sing is not captured in the record, or a programmer's ability to program is not captured in the program, even a program that writes other programs or assembles new programs out of existing ones remains the *product* of a human programmer's ability to program. Programs and records can not capture the abilities of their creators. Records, robots, programs, self-programming and self-organizing systems, refactoring and integration tools, formal mathematical methods, ontologies, the semantic web, all fail to capture their creator's ability to create them. Without a human analyst, they suffer of a severe limitation in their analogy-making capabilities, and are forced to resort to humans every time they need analogies. It has not been possible to fully automate any of them.

It is not possible to engineer our way out of these problems. The technologies require no less than a full automation of the human developer, and are positioned at the very heart of AGI. At the present point, they are still in the realm of mathematics, not engineering, and they require a mathematical solution.

**The present work.** Around 2006, motivated by my interest and expertise in refactoring, and already aware of the canonical representation of systems (see the canonical matrix in Eq. (2) below), but unable to explain why refactoring was so easy for me and so difficult to automate, I decided to find out. I selected an 18-line C program describing a simple problem of Physics [7], scrambled it beyond recognition, prepared the corresponding canonical matrix, and proceeded to refactor the program in small steps while at the same time observing the effect of my decisions on the matrix. After hours of doing the same exercise over and over again, I suddenly noticed that each one of my refactoring decisions led to a symmetric permutation of the matrix that caused the $A$'s to move closer to the diagonal. The average distance of the $A$'s to the diagonal is a *functional* (see §2). As far as I know, this was the first time ever that an observation was made of a high cognitive function of the brain controlled by a mathematical functional.

Soon after that, I was able to simulate the process on a PC, and to carry out a number of computational experiments to make sure that the discovery applied not just to Physics but also to other problem domains, such as object-oriented analysis and design, the recognition of images, refactoring, and parallel programming. It did, all the experiments confirmed the finding. Three of the experiments are discussed in [8], and one more is fully expanded in that publication.

The importance and scope of this discovery can not be over-emphasized. It represents a rigorous mathematical solution, obtained from first principles, not an informed guess based on phenomenology, or an engineering compromise. It does not require any previous intelligence about the problem at hand, just the problem as observed. It is not a computer program, but it can be easily simulated by one and implemented on any computer. It is the core process that finds the

analogies and creates the associations, the organization, and the structures. It is the common root for all the critical technologies of the future just mentioned. This process is called *emergent inference*. Will it allow the full automation of the technologies? I have no doubts it will, because once the abilities are captured, the technologies will follow. But the only way to make sure is to automate them.

The goal of AGI is to design systems capable of human-level intelligence. That statement alone defines the human brain, and more in particular its high cognitive functions, as a reference point to evaluate AGI projects. It also puts natural processes on the map, and raises the question of the origin of intelligence. How can a natural system minimize a functional? All intelligent systems are also dynamical systems, and they all have one thing they try to minimize: their resources. If the functional represents a resource used by the natural system, then the dynamics of the system will minimize the functional and find the analogies, the associations, the organization and the structures which represent the abilities and the intelligence. Which brings us to a very important point. The dynamical process in the natural system is only trying to preserve resources. Its purpose is not to develop abilities or intelligence, and it certainly lacks any intelligence of its own. Intelligence arises as a *side-effect* of a natural process. It can also arise as a side effect of an artificial process.

Think of it a little. It had to be that way. How else can intelligence originate? If intelligence originated only from previous intelligence, then there would have to be something like the *first* intelligence. But where is it? In the DNA? The DNA has a picture of the dancing martian? The only other choice, is that intelligence originates as a side effect from a process that is not trying to originate it.

Section 2 is a mathematical introduction. Section 3 discusses practical aspects and meaning of canonical systems, and the notion of host-guest systems. Section 4 presents a simple but working model of the brain, that demonstrates intelligence as a side effect of a dynamical process that tries preserve resources. Section 5 presents the conclusions.

## 2 Mathematical background

A *knowledge base* is a computational structure that can represent any system of interest, including its dynamics and the changes the dynamics causes on the system, as well as any procedures that affect the knowledge base, their outputs, the procedures that affect the outputs, and so on. The knowledge base must satisfy the property of *closure*. This work proposes a *partially ordered set* as a *universal* knowledge base. A sketch of a proof that a partially ordered set satisfies the requirements can be found in previous publications [8, 9]. Software is particularly easy to convert to partially ordered set format, about the same effort as compiling the code [10]. In addition, partially ordered sets have an unusually rich collection of mathematical properties. For example [8]:

$$\begin{aligned} S &= \{a, b, c, d, e, f, g, h, i, j\} \\ \omega &= \{a < c, b < e, c < j, d < f, e < g, f < h, g < i, h < i, i < j\} \end{aligned} \tag{1}$$

is a partially ordered set, where $S$ is the set and $\omega$ is the partial order, which consists of relations of the form $\alpha < \beta$, or "$\alpha$ precedes $\beta$", where $\alpha, \beta \in S$. The nature or meaning of the elements of $S$ is irrelevant. They can be anything, such as boolean values, functions, computer programs, or entire systems. Or other partially ordered sets. In the example, they represent tasks on a parallel computer.

Let $\pi$ be a legal permutation of $S$. A permutation is said to be *legal* if it does not violate the partial order. A convenient representation of a partially ordered set under a legal permutation $\pi$ is the *canonical matrix*. The following is the canonical matrix for the example of Eq. 1 under permutation $(d, a, b, c, e, f, g, h, i, j)$:

| | $d$ | $a$ | $b$ | $c$ | $e$ | $f$ | $g$ | $h$ | $i$ | $j$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $d$ | $C$ | | | | | | | | | |
| $a$ | | $C$ | | | | | | | | |
| $b$ | | | $C$ | | | | | | | |
| $c$ | | $A$ | | $C$ | | | | | | |
| $e$ | | | $A$ | | $C$ | | | | | |
| $f$ | $A$ | | | | | $C$ | | | | |
| $g$ | | | | | $A$ | | $C$ | | | |
| $h$ | | | | | | $A$ | | $C$ | | |
| $i$ | | | | | | | $A$ | $A$ | $C$ | |
| $j$ | | | | $A$ | | | | | $A$ | $C$ |

(2)

The matrix is square, lower triangular, and sparse [11]. The elements of $S$ are listed as headers for rows and columns. All diagonal elements are marked with $C$, and the relations in $\omega$ are marked with $A$, as follows: if $\alpha < \beta$ is a relation in $\omega$, then element $(\beta, \alpha)$ of the matrix is marked with $A$. The matrix has the important property that all $A$'s are in the lower triangle if and only if $\pi$ is legal.

Let $\Pi$ be the set of all legal permutations of $S$. A *functional* $L(\pi)$, called the *cost* of permutation $\pi$, is now defined over set $\Pi$ as follows: (1) number the elements of $S$ in the order they appear in $\pi$, starting with 1, and let $\nu(\alpha)$ be the number assigned to some element $\alpha \in S$; (2) if $\alpha < \beta$ is a relation in $\omega$, then the cost of that relation is $\nu(\beta) - \nu(\alpha)$; and (3) $L(\pi)$, the cost of permutation $\pi$, is twice the sum of the costs of all relations in $\omega$. A functional is a map that assigns a number, not necessarily different, to each permutation. In this case, the number is a positive integer. For example, permutation $(b, e, g, d, f, h, i, a, c, j)$ is legal. The numbers assigned to elements of $S$ are $\nu(b) = 1$, $\nu(e) = 2$, $\nu(g) = 3$, etc. The cost of this permutation is 28.

The functional plays the role of a *hash* function on the space $\Pi$ of all legal permutations. It partitions $\Pi$ into subsets, each containing permutations with the same cost. It then becomes possible to *minimize* the functional, which means finding the permutation/s with the minimum cost. Minimizing the functional is not always easy, because $\Pi$ may be very large. Montecarlo, or other random search techniques can be used. When the functional $L(\pi)$ is minimized over $\Pi$, a subset of permutations, say $\Pi_{\min} \subset \Pi$, is found at the minimum. Subset $\Pi_{\min}$ has the following remarkable mathematical property:

> Subset $\Pi_{\min}$ is *organized*, and is either a permutation group of set $S$ or a generator for a permutation group of $S$. In either case, it induces a *block system* in set $S$.

A *block system* is a partition of $S$ into *blocks*, or parts, that have the property of being *stable*, meaning they are *invariant* under the action of $\Pi_{\min}$. In other words, the same blocks are found in all the permutations. The block system constitutes *structure*. To each block there corresponds an *equivalence relation*, which simple says that, if $\alpha, \beta \in S$ are two elements of $S$ found in the same block in one of the permutations, they are in the same block in every permutation of $\Pi_{\min}$. The equivalence relations are a new form of inference, called *emergent inference*. They are inference because they are statements of truth, they reveal a new mathematical property that was not explicit in the original data. And they are called "emergent" because the process that finds them is one of *emergence* [12]. It has been proposed that emergent inference is the key to intelligence [8].

It is very, very important to notice that the process that elucidates the emergent inference is completely unrelated to the purpose. The process only searches for permutations with minimum cost. Its purpose is most definitely not to find any structures or logical relations, yet if finds them as a *side effect*. Indications exist that the brain uses a similar mechanism to create structures and generate intelligent behavior, see §4.

But there is more. The block system is itself a set, and it has a partial order induced by the original order $\omega$. A mechanism of *feedback* naturally arises, where structures are repetitively fed back to the search process, and new, more compact and higher-level structures emerge, giving rise to a whole hierarchy of nested structures, such as the 7 level hierarchy shown in the UML diagram of Fig. 1(b) of [8]. Feedback has long been considered as one of the cornerstones of intelligence. In object-oriented programming, the blocks correspond to objects and the hierarchies to inheritance hierarchies.

In the example, the minimum cost is 28, with only 2 permutations:

$$\begin{aligned}&(b,e,g,d,f,h,i,a,c,j)\\&(d,f,h,b,e,g,i,a,c,j)\end{aligned} \tag{3}$$

There are 6 blocks in the block system, 4 of which are trivial. They are $(beg)$ $(dfh)$ $(i)$ $(a)$ $(c)$ $(j)$, which are clearly a stable structure because they appear in both permutations. Blocks $(beg)$ and $(dfh)$ are the inference that has emerged in the first iteration. They correspond to the output of the "intelligent" process. Feedback results in 7 levels of structure, and the progressive build up of intelligence can literally be seen in each one of them. An entire OO analysis and design, obtained with full autonomy, no human intervention. Converting this automatically to a program, say in Java or C#, should be straightforward.

But how does all this apply to our purposes? Recall that emergent inference applies to *all* systems. I can't publish all systems, but I did what I could and I solved and published 4 very different systems: a simple problem of Physics [7], an extensive problem of refactoring and OO design in Java [10], a problem of

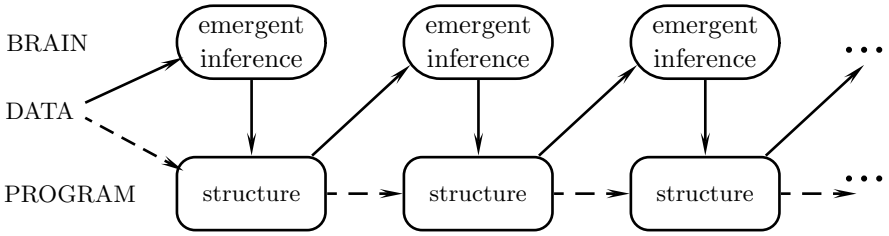


**Fig. 1.** Traditional development cycle for a computer program. Initially, only the solid arrows are installed. A human analyst's brain provides the emergent inference. Frequent human intervention is required. When development is complete, the analyst and the solid arrows are removed, and the dashed arrows are installed. Emergent inference is no longer available, and self-programming will not work. In the environment proposed in this paper, "brain" refers to the artificial emergence inference process. Both sets of arrows are permanently installed, brain and computer are independent processes performed by separate processors, and a self-programming capability remains in place. This diagram is simplified, the real one is a directed graph with many interconnections.

image recognition [13], and a parallel computer [8]. Researchers who want to solve other problems, or revisit the ones I have solved, are welcomed.

## 3 Canonical systems and the host-guest concept

A *canonical system* is one where knowledge is represented as a partially ordered set, and an ongoing dynamical process exists, either natural or artificial, that constantly minimizes the functional and produces organized structures by emergent inference. The system is part of its environment and learns from it, and tries to adapt by constantly reprogramming itself, just like Ikon Flux. Unlike Ikon Flux, the canonical system is self-organizing, self-programming, and naturally integrated.

The input to the canonical system consists of *teacher instructions* (see §IV of [10]), provided by a teacher or obtained from sensor input, that contains references to structures already existing in the canonical system (hence the need for a bootleg sector), and is equivalent to one or several relations in the partial order. As the system learns and forms more advanced structures, these can in turn be referenced in the instructions, and the learning process becomes more powerful. The situation very strongly reminds of an infant that begins by learning from its senses, grows into a child that receives instruction from a school teacher, and into a college student that assimilates advanced knowledge. The structures that the canonical system generates are the new programs.

The nested hierarchy is an *ontology* [14], but not just for the domain of interest but also for particular objects in it (for example, an image in an image recognition domain). Ontology generation in the canonical system is natural. Just as any ontology, the canonical system can be used from top-down as an associative memory, and from bottom-up to match patterns.

Is the canonical system a *complete* AGI system? I believe it is. Here is the argument that supports my conjecture: (1) any system can be represented as a partially ordered set, and (2) if any additional system is needed for AGI completeness, that system can be represented as a partially ordered set and integrated into the original one. There are many more things that can be done with canonical systems, for example several areas where they can easily outperform humans have been identified and published in the literature. But this paper must remain focused on its original goals.

The host-guest concept [9] captures the fact that two different, well-differentiated, and independent processes exist in a self-programming system. One is the dynamical process that minimizes the functional and causes the inference to emerge as a side effect. This process is equivalent to refactoring and does not change the program or affect execution. It can run concurrently in the background or on a separate processor. Figure 1 illustrates the concept.

The other process is the actual execution of the program. This process relies on the functionality stored in the connections of the canonical system, not on its organization or structures, and can run even while the connections are being reset, provided of course the connections do not change.

The next Section introduces a simple model of the brain and demonstrates how emergent inference arises as a side effect of resource preservation.

## 4 A viable brain model

A *model* is an abstraction of a real-world system where only some features of interest are captured and the rest are ignored. Each model has a *purpose*, which determines how the abstraction is made, and a *scope*, which is the domain of problems the model can solve. The model does not have to resemble the real-world system. All it has to do is to recognize the features of interest and produce results that agree with observation.

The purpose of the present model is to demonstrate how emergent inference works in the brain. The model consists of a set of interconnected neurons. Each neuron has only one capability: it fires when its upstream neurons fire. And each connection also has only one capability: it pulls from the neurons to make them come closer and save resources. The partially ordered set for this model is straightforward. $S$ is the set of neurons, and $\omega$ is the order of firing. If neuron $D$ fires only when its upstream neurons $A$, $B$ and $C$ fire, then $A < D$, $B < D$ and $C < D$ are relations in $\omega$. This model is not a neural network, there are no weights associated with the connections, and no feedback algorithms.

To interpret the model, I'll show next how it relates to what was said in §2 and §3. First, since the nature of the elements of $S$ is irrelevant (it doesn't matter that they are neurons), the only place where information can be stored is the partial order $\omega$. It follows that $\omega$ is the *memory*. Knowledge is stored in the connections, something that neuroscientists have been saying for years. A relation in $\omega$, which involves 2 elements, corresponds to a connection in the model, which involves 2 neurons. The only property of the relation, its cost,

corresponds to the only property of a connection, its length. The partial order $\omega$, which involves all elements, corresponds to the network, which involves all neurons, and the permutation of $S$ corresponds to the physical arrangement of the neurons. Two neurons that exchange their positions to save resources represent a step in the search for another permutation with a lower cost. The network stabilizes only when the total length of all connections is minimized, which is precisely where the functional attains its minimum value.

Under this dynamics, groups of neurons with many connections will come closer together, while pushing neurons with fewer connections further apart. Physical clustering takes place. Clusters of strongly interconnected, weakly coupled neurons physically form. This behavior has in fact been experimentally confirmed [15]. The clusters are called *neural cliques*, and they have been experimentally observed and characterized in the brain, thus providing the strongest confirmation yet that emergent inference actually happens in the brain.

Once the neural cliques have formed, the process continues, but now the "neurons" are the neural cliques. Inter-click connections pull from the cliques, causing them to cluster into higher order cliques. This is a process of natural feedback, and it continues until the entire organ is formed. There is no experimental verification for the feedback. The brain is a very simple machine. Contrary to what David Eagleman says [16], the brain is very simple, but it is still powerful enough for us to understand it.

## 5 Concluding remarks

This paper has proposed the argument that computer programs are written by human developers, and that a full self-programming capability requires no less than a full automation of the human developer's high brain functions. The paper has also argued that, by the definition of AGI, such an automation can be achieved only by a fully capable AGI system. This paper has further argued that the current engineering approach to self-programming, illustrated in Fig. 1, does nothing but to shift the problems caused by the absence of the human developer from one place to another. The approach will inevitably reach a point where refactoring and integration must be applied, and further automation will become impossible beyond that point. Refactoring and integration are the two critical technologies where the absence of the human developer becomes immediately manifest, and they have not been automated yet.

To address these limitations, and based on a reported experimental observation of a high cognitive function of the human brain being controlled by a mathematical functional, the paper has also proposed that the recently announced theory of emergence should be applied. In the theory, a partially order set is used as the knowledge base, and a mathematical functional is defined, the minimization of which by an unrelated dynamical process originates a new, recently discovered type of logical inference, called emergent inference. Emergent inference finds analogies in the available knowledge, and uses the analogies to organize the knowledge into structures, an entire ontology. This process is equivalent to

refactoring, and results in a system that is naturally self-organized, naturally self-programming, and naturally integrated. This system is also proposed as the fundamental AGI system. To support the findings, several computational experiments are described, and a simple but functional model of the brain is proposed.

The conclusions reached in this paper are easy to implement in practice on any computer. The implementations may or may not be very efficient, though, unless until suitable self-organizing hardware is developed, perhaps following the general ideas proposed for the brain model of Section 4.

## References

1. Pissanetzky, S. Adaptive systems and analyst-independent technologies. Workshop on Automation and Robotics. NASA Gilruth Center, Johnson Space Center, Nov. 12, 2010. Houston, Texas, USA. Summary available from author's web site.
2. Nivel, E., Thórisson, K. R. Self-Programming: Operationalizing Autonomy. Proc. 2nd. Conf. On Artificial General Intelligence, Atlantis Press (2009).
3. Opdyke, W. F. Refactoring object-oriented frameworks. Ph.D. thesis, Dep. Comp. Science, Univ. of Illinois, Urbana-Champaign (1992).
4. Garrido, A.,Johnson, R. Challenges of refactoring C programs. Proc. International Workshop on Principles of Software Evolution, Orlando, Florida. 6-14 (2002).
5. Wilson,G. Refactoring the law: reformulating legal ontologies. Juris Dr. Writing Requirement, School of Law, Univ. of San Francisco (2006).
6. Chan, L. Y., Kosuri,S., Endy, D. Refactoring bacteriophage T7. Molecular Systems Biology, article number: 2005.0018. Published online: 13 September 2005.
7. Pissanetzky, S. Applications of the Matrix Model of Computation. Proc. 12th. WMSCI Conference IV, 190-195 (2008).
8. Pissanetzky, S. Structural emergence in partially ordered sets is the key to intelligence. To appear in Lecture Notes in Artificial Intelligence, a subseries of Lecture Notes in Computer Science, Springer-Verlag. Publication is scheduled for August 2011. The original publication is available at www.springerlink.com. Currently available from author's web site.
9. Pissanetzky, S. Coupled Dynamics in Host-Guest Complex Systems Duplicates Emergent Behavior in the Brain. World Academy of Science, Engineering and Technology, 68, 1-9 (2010)
10. Pissanetzky, S. A new Universal Model of Computation and its Contribution to Learning, Intelligence, Parallelism, Ontologies, Refactoring, and the Sharing of Resources. Int. J. of Information and Mathematical Sciences 5, 143-173 (2009)
11. Pissanetzky, S. Sparse Matrix Technology. Academic Press (1984).
12. Prokopenko, M., Boschetti, F., Ryan, A. J. An Information-Theoretic Primer on Complexity, Self-Organization, and Emergence. Complexity 15, 11-28 (2009)
13. Pissanetzky, S. A new Type of Structured Artificial Neural Networks based on the Matrix Model of Computation. In: Arabnia, H. R., Mun, Y., (eds.) The 2008 International Conference on Artificial Intelligence I, 251-257 (2008)
14. Chandrasekaran, B., Josephson, R., and Benjamins, V. R. What are ontologies, and why do we need them? IEEE Intelligent Systems, pp. 20-26, Jan/Feb 1999.
15. Lina, L., Osana, R., Tsiena, J. Z. Organizing principles of real-time memory encoding: neural clique assemblies and universal neural codes. Trends in Neurosciences, 29, 48-57 (2006).
16. Eagleman, D. Incognito: The secret lives of the brain. Pantheon Press, 2011.