Cognitive Computation (2016) 8:555
The final publication is available at Springer
via http://dx.doi.org/10.1007/s12559-015-9378-0

Learning the semantics of notational systems with a semiotic cognitive automaton

Valerio Targon, European Patent Office
The Hague, The Netherlands

Abstract Through semiotic modelling, a system can retrieve and manipulate its own representational formats to interpret a series of observations; this is in contrast to information processing approaches that require representational formats to be specified beforehand and thus limit the semantic properties that the system can experience. Our semiotic cognitive automaton is driven only by the observations it makes and therefore operates based on grounded symbols.

A best-case scenario for our automaton involves observations that are univocally interpreted - i.e., distinct observation symbols - and that make reference to a reality characterised by "hard constraints". Arithmetic offers such a scenario. The gap between syntax and semantics is also subtle in the case of calculations.

Our automaton starts without any a priori knowledge of mathematical formalisms and not only learns the syntactical rules by which arithmetic operations are solved but also reveals the true meaning of numbers by means of second-order reasoning.

Keywords symbol grounding problem · semiotic modelling · second-order reasoning · extended correlation · desmogram · abductive reasoning

1 Introduction

A controversy has arisen in cognitive science concerning the possible ability of a symbol system to reason and understand. The physical symbol system hypothesis, attributed to Newell and Simon [1], states that "a physical symbol system [such as a digital computer, for example] has the necessary and sufficient means for intelligent actions", implying that a symbol-processing program could make a computer capable of understanding (understanding represents a prerequisite for acting intelligently and is the scope of this article).

The views expressed in this article reflect my personal opinion and not that of the EPO

E-mail: valerio.targon@asp-poli.it

Assuming, in an artificial intelligence program, operating according to some formal semantics, far more understanding and causality than is warranted is a fallacy of the human mind, which has been coined the "ELIZA effect" [2].

Searle [3] considers that computers cannot be said to understand because they manipulate formal, meaningless symbols. These symbols exist only in a conventional relationship with their referents, i.e., their meanings are parasitic in the minds of the programmer and of the interpreter of the computer output, where they stand for objects in the world. To challenge the assumption that a symbol system that exhibits some intelligent behaviour actually understands its symbols and its own manipulation of them, Searle proposes the thought experiment of the Chinese room to illustrate that taking in symbols, processing these symbols based on rules, and outputting other symbols as directed by said rules and in accordance with the input tokens does not require any understanding of the meaning of the symbols by the computer. Objects in the world are beyond the scope of the computer because "programs are not machines", i.e., they are not subject to causality and cannot perform any actions.

Harnad [4] extends Searle's analysis by introducing the symbol grounding problem, according to which an intelligent agent should own the meaning of the symbols with which it operates (otherwise, no imagined action could be actually executed by the agent). Harnad proposes to ground symbolic representations in behavioural interactions with the environment, thereby emphasising the role of embodiment, and through the use of artificial neural networks (more recently, sub-symbolic/symbolic neural networks have been advocated [5]). This proposal amounts to a theory of "bottom-up" symbol grounding, requiring sensorimotor capabilities, direct experience of the world and the ability to associate symbols to sub-symbolic processes, e.g., through categorical representation [4] (i.e., a program is insufficient). However, nothing warrants the conclusion that symbols thus grounded in sensory perception of reality hold an intrinsic meaning for the system.

In the following, we posit the symbol grounding problem in its most general relation to the problem of meaning: how symbols inside an agent autonomously obtain their meaning and what meaning is. Even without any interaction with the environment, a program can be built around "symbolic experience" [6], which cannot be reduced to "dictionary-go-round" [4] but rather is equivalent to "getting meaning out of meaninglessness".

Computers, such as Searle's Chinese room, accept input icons and generate output icons. A system that is acting purely as a syntactic manipulator of icons or icon sequences cannot achieve intelligence, because "syntax is not sufficient for semantics" [3]. Let us consider a pocket calculator: it is a symbol system that behaves in an intelligent manner but holds no understanding of the symbols it manipulates, which are extrinsic, ungrounded and meaningless to it [7]. It cannot relate the 7-key or the 7-display to a concept of 7, although it has an internal representation of numbers, including 7. The understanding of these icons as representations and their meaningful interpretation in a given context require acts of human semiosis [8]. An interpretation occurs when an internal representation of an object is evoked, starting from input data, or from a previously identified representation. The source of information originating the interpretation is called a *sign* and performing semiosis means extracting meanings. A symbol system such as the physical symbol system described by Newell and Simon [1] is not a semiotic system [9]. A program following a sequence of instructions does not have any possibility of escaping the internalist trap [10], but an automaton following a certain sequence of instructions can simulate semiosis [11].

Artificial semiosis should not be restricted only to machines with robotic capabilities, such as Meystel's "semiotic automaton" [12] or, more recently, the robots proposed by Vogt that interact in language games and construct "semiotic symbols" from scratch [13].

We propose an automaton that is capable, with no a priori knowledge, of receiving symbols as inputs and elaborating upon them to produce an evolving collection of *semiotic symbols* as an output. We suggest that symbols of formal notational systems (e.g., words in a language, numbers in mathematics) can receive meaning through being grounded in internal representations, each of them just a portion of the large internal reality of an agent, with said meaning residing in the evolving relations among the fragments of this inner world. First, the automaton creates from scratch certain representational structures (i.e., semiotic symbols, which are intrinsically grounded as a result of their semiotic definition; see the following section). Then, to provide the automaton with an experience, we require it to have inductive (grounded) access to certain abstract concepts, such as time, which cannot be sensed directly through the senses, and quantity, which is independent of any concrete object. Certain of the symbol structures generated are then discovered, through an inductive learning process, to be capable of predicting and explaining a process observed by the automaton. These symbols receive further meaning as the cognitive automaton connects them to a portion of its internal reality, i.e., to internal and autonomously discovered constraints, quantified through successful prediction attempts over time. We say that these *semiotic symbols* are *grounded in a cognitive manner*. The automaton's process of symbol generation should be continuous and overproducing (only a fraction of the generated symbols may receive further meaning during processing). A method of unsupervised, cumulative symbol generation is offered by *semiotic modelling* [14]. Semiotic cognitive grounding is possible in all situations in which feedback control can be employed in connectionist learning and artificial neural networks, although with both a qualitative and a quantitative difference. The qualitative difference is that neural networks are based on nonsymbolic functions and can in no way be interpreted through semantics. Furthermore, cognitive grounding is not limited to situations that require the matching of an input with an output but rather can address a multitude of qualitatively different processes and can focus, for example, on the synthetic power of expression of the symbols or on their hierarchical organisation.

Understanding can be verified - beyond subjectivity - through behavioural tests demonstrating whether an agent can recognise, describe and use all of the entities to which its symbols refer (and, at a high level of abstraction, whether it can perform analysis, synthesis and evaluation of said entities, as done by human learners, according to Bloom's taxonomy of the cognitive domain [15]). Our prototype semiotic cognitive automaton is faced with the task of learning arithmetic in response to a specific input, i.e., number sentences in the form of arithmetic operations. Considering the case of arithmetic operations simplifies the input model for the automaton: there exists simply a set of distinct symbols, and no categorical representation [4] is needed. The automaton is not programmed to perform arithmetic operations and does not implement an algorithm targeted to learn arithmetic; it is a system initialised with no a priori knowledge that solves the problem of learning per se. It receives symbols comprising ciphers and operational signs and employs the regularities of arithmetic operations to organise these symbols and construct higher-level ones until it learns how to solve any arithmetic operation, turning itself into a "semiotic" pocket calculator. By contrast, compare our automaton to an artificial neural network solving arithmetic operations (one could have a design for such a neural network that provides an exact solution [16], or one could design a feed-forward neural network and train it to learn arithmetic operations [17]). Neural networks are typically referred to as black boxes. How can we assess the level of understanding attained by our artificial learning system instead? It could pass a test in the form "`153+298=`", but what about a subtler one in the form "`4;7;10;13;`"?

A "semiotic" pocket calculator seems to capture the semantic notion of calculation solely in terms of syntactical symbol manipulation, i.e., typographical rules operating on strings of symbols, without any connection to the meanings of the numbers (see also Hofstadter's specification of a syntactical pocket calculator [18]). However, semiotic cognitive grounding also enables *second-order* reasoning, such that the automaton assigns another meaning to the symbols it receives and manipulates. Such meanings, by which the symbols become more than simple placeholders, are based on the ability of the automaton to observe and reason about its own structure. The automaton can establish a connection between the symbols of its input and its internal processes and representations, which it must consider to be as real as the input it observes. In the case of the "semiotic" pocket calculator, this connection is established between the ciphers and an ordering that the automaton has constructed, and it reveals the meaning of the ciphers. It is then possible for the automaton to interpret external structures and symbols based on its own inner structuredness. One might be tempted to claim that learning arithmetic is possible because the automaton already possesses an internal mathematical reality, being based on an algorithm; however, second-order reasoning is instead grounded in the abstract concepts of time and quantity, which are inductively available to the automaton (making it, in this sense, more a machine than a program), and representations of objects, categories and concepts may all equally become subjects of second-order reasoning.

Recent work in cognitive computing has begun to address the theoretical possibility that a machine can attain a conscious state (based on non-abstract properties of its physical system implementation [19]) and that it can use its input/output observations to learn a model of itself (based on a machine learning algorithm with strong learning biases [20]), capabilities that clearly go beyond the level of brute input and output, i.e., beyond first order.

The proposed semiotic cognitive automaton could be applied to any problem involving a codified input, the most notable example being an unstructured or partially structured text. Whereas the theory of "bottom-up" symbol grounding requires a sensory interface to connect the system to the outside world, semiotic cognitive grounding is not subject to any such requirement for the assignment of meaning to symbols (in particular, the meaning of numbers is derived solely from their relations to one another, without reference to physical quantities, and arithmetic operations are regarded as actions taken on numbers that result in changes to these numbers, where the numbers being changed are purely conceptual entities). It would be incorrect to assert that the semiotic cognitive automaton is *disembodied*; this term should be reserved for traditional artificial intelligence systems that use "symbols" as labels for objects in the outside world and treat them in a manner independent of any experience, solely in accordance with formal rules [21]. The semiotic cognitive automaton behaves in accordance with its experience, i.e., it uses - subject to its limitations - all of the information it receives as inputs for processing. To it, the meanings of symbols reside in the relations among them, rather than being assigned arbitrarily by an outside observer. However, because of the fundamental difference between the perspectives of an embodied agent (e.g., a robot, or even a human, interacting with the environment) and of an automaton such as ours, the reasoning performed by the two types of systems about the same input may often differ.

2 The structure of the semiotic cognitive automaton

At the beginning of the learning process, the semiotic cognitive automaton is capable only of recognising base symbols, i.e., it is able to discriminate that a given symbol (e.g., perhaps an ASCII character, but also perhaps an indicator of a value on a multi-level scale, or the suit and rank of one of a set of playing cards, or a Chinese ideogram), and not another (ASCII character, or indicator, or playing card, or Chinese ideogram, respectively), occurs at certain positions in an input. We do not associate any predefined representations with the input symbols but rather have the automaton perform syntagmatic and paradigmatic analysis of the input. These terms are drawn from the field of linguistics; in language, following de Saussure, meaning is produced through paradigmatic and syntagmatic systems [22]. For example, in a sentence "a dog bites a man", "dog" is chosen from among a number of words, such as "spider", "snake", and so on, that could have filled the same slot based on the paradigmatic system, i.e., could be substituted one for another without disturbing the syntax of the sentence. Moreover, both the sentences "a dog bites a man" and "a man bites a dog" consist of the same words. However, the meanings of the two sentences are different because the words that compose the sentences are arranged differently based on the syntagmatic system, i.e., their words are combined in different specific orders in accordance with grammatical rules. More generally, (i) syntagmatic analysis refers to a synchronic perspective and is concerned with “horizontal relations” based on combination, e.g., concatenation and co-occurrence; (ii) paradigmatic analysis refers to a diachronic perspective and is concerned with “vertical relations”, i.e., paradigmatic alternatives governed by the principle of selection, or choice.

Our automaton

– extracts syntagmatic relations from its input,
– reveals the paradigms of the syntagms, and
– implements an iterative procedure: once a paradigm is found, it becomes a new symbol on which syntagmatic and paradigmatic analysis is performed.

The automaton then continually creates internal constraints to explain the coexistence of the symbols that it generates such that its inner world grows in abundance.

2.1 Relations managed by the semiotic cognitive automaton

The semiotic cognitive automaton receives a concatenation of discrete symbols as an input. These symbols may be concatenated in space or in time; there is an effective difference between the two only if we assume that the automaton has finite memory. Let us suppose that the automaton has access to its entire input at any time. In analogy with computational linguistics, we refer to the input as the *corpus*.

Let $\mathcal{A}$ denote the alphabet that contains all symbols occurring in the corpus. For each symbol $i \in \mathcal{A}$, the empirical probability of occurrence $p(i)$ can be computed as follows:

$$p(i) = \frac{\#(i)}{N}, \tag{1}$$

where $\#(i)$ returns the number of occurrences of the specified symbol over the entire corpus and N is the size of the corpus in symbols.

Any two adjacent symbols in the corpus exist in a binary relationship of precedence/subsequence. The number of occurrences of symbol i immediately preceding symbol j - or, equivalently, the number of occurrences of symbol j immediately following symbol i - is denoted by $\#(i, j)$, where (i, j) represents a digram. n-grams extend the notion of a digram.

However, n-grams can represent only a limited set of relations. To achieve greater freedom in analysing relations in the corpus, we adopt *regular expressions* instead of n-grams. n-grams can be regarded as a special case of regular expressions, namely, literals. We use conventional Perl notation for regular expressions [23]. In Perl notation, a digram is represented as $\boxed{\text{ij}}$. Additional relations that can be represented include exact k-hop precedence, which is denoted by $\boxed{i\texttt{.\{k\}}j}$, with the value k enclosed in curly brackets, and maximal k-hop precedence, which is denoted by $\boxed{i\texttt{.\{0,k\}}j}$.

Given a regular expression g, $\#(g)$ can be computed either as the number of overlapping matches in the corpus or as the number of non-overlapping leftmost shortest matches. Different rules give rise to different computations of the occurrence probability $p(g)$. In the following, we employ the non-overlapping leftmost shortest-match rule.

Before computing the number of occurrences of a given regular expression, its a priori probability of occurrence can be derived through deductive reasoning if there exist two regular expressions, f and g, whose empirical probabilities of occurrence are known, such that they produce the original regular expression once concatenated. We denote the original regular expression by $f \diamond g$, where the symbol $\diamond$ represents the operator of concatenation. The a priori probability of occurrence of such a concatenation can be derived as follows:

$$p_0(f \diamond g) = p(f)p(g). \tag{2}$$

In the same manner, the maximum value of the probability of occurrence of the concatenation can be found as follows:

$$p_{max}(f \diamond g) = \min\{p(f), p(g)\}. \tag{3}$$

Thus, for example, $p_0(\boxed{ij}) = p(i)p(j)$ and $p_{\max} = \min\{p(i), p(j)\}$, with $i, j \in \mathcal{A}$.

Regular expressions can also contain groups, which are indicated in parentheses in Perl notation. A group may contain literals or, more generally, any regular expressions. If the regular expressions $f_1, f_2, \ldots, f_n$ that form a group are all distinct such that none is a superset of any other, it holds that

$$p(\boxed{\texttt{(}f_1\texttt{|}f_2\texttt{|}\ldots\texttt{|}f_n\texttt{)}}) = p(f_1) + p(f_2) + \cdots + p(f_n). \tag{4}$$

Groups are equivalent to paradigms, as they are governed by the principle of *selection*. Suppose that the empirical probability of a regular expression $e \diamond \boxed{\texttt{(}f_1\texttt{|}f_2\texttt{|}\ldots\texttt{|}f_n\texttt{)}} \diamond g$ that contains a group of regular expressions has been computed. One could derive the a priori probability of the more specific regular expression that matches only the regular expression f_i of this group (thereby selecting f_i) as follows:

$$p_0(e \diamond f_i \diamond g) = \frac{p(e \diamond \boxed{\texttt{(}f_1\texttt{|}f_2\texttt{|}\ldots\texttt{|}f_n\texttt{)}} \diamond g)p(f_i)}{p(\boxed{\texttt{(}f_1\texttt{|}f_2\texttt{|}\ldots\texttt{|}f_n\texttt{)}})}. \tag{5}$$

Concatenation and selection are the fundamental types of relations that the semiotic cognitive automaton can identify and match in the corpus. Each match causes the number of occurrences of a given relation to be incremented by one unit. We also require the automaton to compute prior probabilities for these relations.

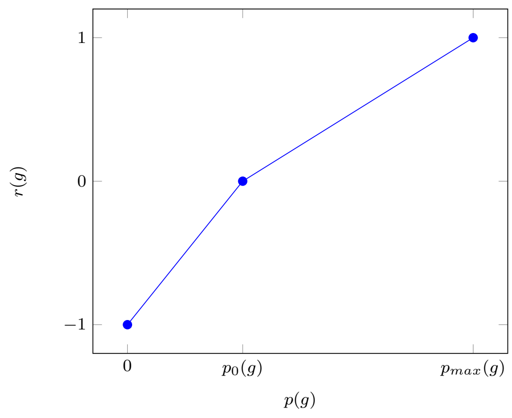

Fig. 1 Relevance r of a composite syntagm g as a function of its occurrence probability p.

2.2 Components of the semiotic cognitive automaton

The semiotic cognitive automaton comprises two main components: the syntagmatic algorithm and the paradigmatic algorithm.

The syntagmatic algorithm. The syntagmatic algorithm is able to compute the a priori probabilities and empirical probabilities of occurrence of linear combinations of symbols, or syntagms, as described in Section 2.1. Moreover, because the number of syntagms existing in the corpus is potentially unbounded, the algorithm is tasked with selecting and delivering to the paradigmatic algorithm a subset of the possible syntagms, such that the number of relations processed by the paradigmatic algorithm remains low.

The syntagmatic algorithm selects only syntagms that carry additional information about the corpus. Note that in any syntagm, one or more hypotheses are implicit. For example, once we know the number of occurrences in the corpus of the syntagm $\boxed{ij}$, namely, $\#(\boxed{ij})$, it is natural to posit that after a symbol i, a symbol j will be found with probability $\#(\boxed{ij})/\#(i) \in [0, 1]$. The probability invoked in this hypothesis can differ significantly from $p(j)$. We can then select syntagms with associated hypotheses that better describe and explain the corpus. Such a criterion will minimise the computational requirements of the paradigmatic algorithm and will be acceptable provided that upon execution of the paradigmatic algorithm using these syntagms, paradigms are created that can form new syntagms (syntagms of a superior level) that also satisfy the criterion. In Section 3, we demonstrate that the semiotic cognitive automaton can produce several iterations of *semiotic modelling* using the following criterion for selecting among syntagms:

$$p(g) - p_0(g) > \mathrm{Th}, \tag{6}$$

where we recall that we treat syntagms as regular expressions and where $\mathrm{Th} > 0$ is an appropriate acceptance threshold.

Note that the difference between $p(g)$ and $p_0(g)$ takes values on the interval $[-p_0(g), p_{max}(g) - p_0(g)]$. This dependence of the domain makes it difficult to define a common threshold for syntagms with different maximum values of their probability of occurrence. One possible approach is to define a new metric, namely, the order-2 relevance of the syntagm, to be bounded on $[-1, 1]$. Let us compute this quantity as follows:

$$r(g) = \begin{cases} \frac{p(g)-p_0(g)}{p_{max}(g)-p_0(g)}, & \text{if } p(g) \geq p_0(g), \\ \frac{p(g)-p_0(g)}{p_0(g)}, & \text{otherwise}, \end{cases} \tag{7}$$

where we use a spline of order 2 - i.e., linear interpolation - to connect the points $(p(g) = 0, r(g) = -1)$, $(p(g) = p_0(g), r(g) = 0)$ and $(p(g) = p_{max}(g), r(g) = 1)$, as shown in Figure 1.

Clearly, $r(g) = 0$ implies that the syntagm g - and the information regarding its frequency in the corpus - is irrelevant given the available prior knowledge. Our selection criterion is thus rewritten as

$$r(g) > \text{Th} \tag{8}$$

and represents a suitable alternative to statistical significance testing.

Because we have established a connection between syntagm selection and hypothesis selection, the entire process of "hypothesis generation", "hypothesis evaluation" and "hypothesis testing" requires our attention. Such a three-phase process has been connected to the three forms of reasoning classified by Charles Sander Peirce [24]. Table 1 introduces the Peircean categories of abduction, deduction and induction. They can serve as a model for the syntagmatic algorithm of the semiotic cognitive automaton. One requirement for the automaton to act as an artificial cognitive system is therefore the implementation of automated abductive, deductive and inductive reasoning [25].

Peircean form of reasoning	Cognitive interpretation [25]	Implementation in the semiotic cognitive automaton
Abduction	(1) provide possible explanations for collected information (hypothesis generation)	several methods of creating new syntagms
Deduction	(2) deduce observable consequences from hypotheses	computation of $p_0(\cdot), p_{max}(\cdot)$
Induction	(3) test consequences of hypotheses	$r(\cdot) > \text{Th}$? (cfr. Inequality 8)

Table 1 Peirce's model of the scientific method, its cognitive interpretations and our corresponding implementations in the semiotic cognitive automaton.

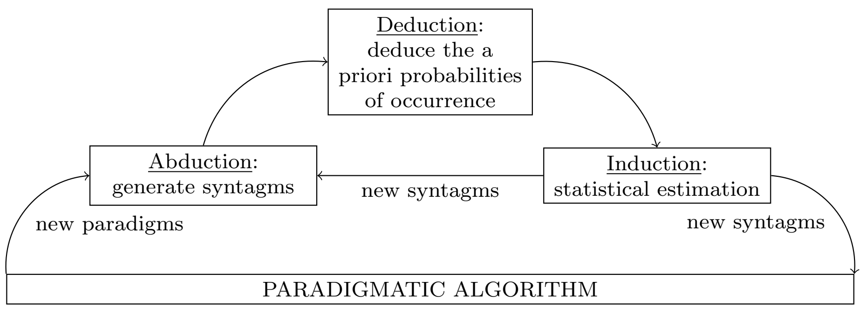

Fig. 2 The (simple) reasoning loop of the syntagmatic algorithm and its relation to the paradigmatic algorithm.

The semiotic cognitive automaton is constantly faced with facts in need of interpretation. It employs semiotic modelling to construct hypothetical representations, i.e., syntagms and paradigms [14]. We have shown how it is possible to generate hypotheses from syntagms. Syntagms themselves can be generated in several simple ways: (i) through random concatenations including new paradigms, (ii) through the concatenation of partially overlapping existing syntagms, (i) through selection among paradigms in existing syntagms, and (iv) by analogy with existing syntagms. We note the following:

– Using any of the aforementioned methods, a new syntagm is generated. *Abduction* occurs. The generated syntagm represents a hypothesis.
– Computing the a priori probability and the maximum probability value of the syntagm corresponds to evaluating consequences of the hypothesis and requires *deduction.*
– *Induction* enables the computation of the empirical probability of the syntagm and the testing of whether the consequence of the hypothesis holds true.

The proposed automated reasoning loop is so simple that there is only one possible consequence for each hypothesis. Syntagms are accepted or rejected on the basis of the difference between their empirical and a priori probabilities following the selection criterion of Eq. 8.

Figure 2 illustrates the closed-loop reasoning process for the semiotic cognitive automaton. The syntagmatic algorithm must return new syntagms - each with an associated metric of relevance - on which abductive reasoning is then performed once again or which are then used by the paradigmatic algorithm.

The syntagmatic algorithm reveals explanatory rules or constraints concerning the input, also making use of symbols autonomously created by the automaton, i.e., by the paradigmatic algorithm. We use the term *semiotic cognitive grounding* (see the Introduction) to refer to the process of connecting lower-level symbols to syntagms that express rules of induction.

The paradigmatic algorithm. The results of induction are used by the *paradigmatic algorithm* to build paradigms. For convenience of representation, let us introduce regular expressions that contain variables as matching expressions. In Perl notation [23], variable names are introduced by the special character $\boxed{\texttt{\$}}$. For example, let

$$f_{digram}(\boxed{\texttt{\$var1}},\boxed{\texttt{\$var2}}) = \boxed{\texttt{\$var1\$var2}} \tag{9}$$

indicate any possible sequence of values that can be assigned to the two variables $\boxed{\texttt{\$var1}}$ and $\boxed{\texttt{\$var2}}$.

Paradigms involve a selection operation among alternatives. The principle of selection operates in a top-down manner in the sense that the alternatives emerge once a specific paradigm has been identified. Paradigms can be identified by means of a “commutation test” [26]. In this test, aspects of the signifier are modified to a certain degree to verify whether a modification of the signified occurs as a result. Clearly, top-down paradigmatic analysis cannot be performed by an unsupervised automaton. Conversely, our automaton builds paradigms in a bottom-up fashion, making use of syntagmatic relations. To date, the lack of a reliable, versatile mechanism for performing bottom-up paradigmatic analysis has limited the possibilities of semiotic modelling.

Paradigmatic abstraction based on the Pearson correlation coefficient has previously been proposed [27]. The Pearson correlation coefficient returns values on the interval $[-1, 1]$. It computes the correlation between two variables of which samples are available, and it represents a normalised covariance. It is also known as the product-moment correlation, and it is typically denoted by ρ. Let us suppose that the corpus is divided into K distinct collections of symbols. For example, the corpus may consist of a set of text articles, as in [27]. The division of the corpus into collections must follow appropriate logic, and these collections should be homogeneous. Let $p_k(i) = \frac{\#(i)}{N_k}$ represent the probability of occurrence of symbol i in the k-th collection, which is N_k symbols in size. Let P_i be the variable associated with the occurrence probability of symbol i in the generic collection. Each collection k serves as a statistical sample, on which P_i is evaluated as $p_k(i)$. The expected value of P_i is clearly $p(i)$. The correlation between P_i and P_j is then estimated as follows:

$$\rho(P_i, P_j) = \frac{\sum_{k=1}^{K}(p_k(i) - p(i))(p_k(j) - p(j))}{\sqrt{\sum_{k=1}^{K}(p_k(i) - p(i))^2}\sqrt{\sum_{k=1}^{K}(p_k(j) - p(j))^2}}. \tag{10}$$

The expected value of the occurrence probability in any subcomponent of the corpus is, indeed, $p(i)$.

By extension, we say that the correlation between two symbols is the correlation between their occurrence probabilities. We note that occurrence in a logical subcomponent of the corpus can be regarded as an example of a syntagmatic relation. Another example of a syntagmatic relation is concatenation with a certain other symbol, and the total number of possible relations is infinite. The potential of this extended correlation has not previously been examined.[1]

[1] I learned of this use of the extended correlation from the Italian computer scientist Piero Slocovich.

We know from Section 2.1 that any syntagm is associated with a given a priori probability of occurrence. Let us consider a finite set of syntagmatic relations and define the vector of differences as follows:

$$\boldsymbol{\delta}_i = \begin{bmatrix} p(f_1(i)) - p_0(f_1(i)) \\ \vdots \\ p(f_Z(i)) - p_0(f_Z(i)) \end{bmatrix}, \tag{11}$$

where $f_1(i), \cdots, f_Z(i)$ represent Z different regular expressions that contain the symbol i as a variable, i.e., Z different syntagms involving symbol i.

We can construct a vector $\boldsymbol{\delta}_i$ such that it has an average of zero. For example, by denoting the L base symbols in the corpus by $a_1, \cdots, a_L$ and choosing $f_1(i) = \boxed{ia_1}, \cdots, f_L(i) = \boxed{ia_L}$, we obtain the following vector:

$$\boldsymbol{\delta}_i = \begin{bmatrix} p(i \diamond a_1) - p(i)p(a_1) \\ \vdots \\ p(i \diamond a_L) - p(i)p(a_L) \end{bmatrix}, \tag{12}$$

which satisfies the constraint $\text{mean}(\boldsymbol{\delta}_i) = 0$. In the following, when computing the correlation coefficient, we do not impose such a constraint on the average of the vector of differences.

The semiotic automaton computes the correlation coefficient of any two symbols i and j as follows:

$$\rho(i,j) = \frac{\boldsymbol{\delta}_i^T \boldsymbol{\delta}_j}{\|\boldsymbol{\delta}_i\| \|\boldsymbol{\delta}_j\|}, \tag{13}$$

where $\|\mathbf{y}\|$ represents the ℓ^2-norm of $\mathbf{y}$.

Let us further generalise the correlation. First, we can use our relevance metric in $\boldsymbol{\delta}_i$ in place of the difference in probability when considering syntagms with different maximum values of their probability of occurrence. Second, the syntagms in $\boldsymbol{\delta}_i$ may be associated with different degrees of importance. This can be expressed by a weight vector $\mathbf{w}$ with a size Z equal to that of $\boldsymbol{\delta}_i$. Let us introduce the weight matrix $\mathbf{W} = \text{diag}(\mathbf{w})$. Then, the correlation becomes

$$\rho(i,j) = \frac{\boldsymbol{\delta}_i^T \mathbf{W} \boldsymbol{\delta}_j}{\|\boldsymbol{\delta}_i\|_{\mathbf{w}} \|\boldsymbol{\delta}_j\|_{\mathbf{w}}}, \tag{14}$$

where $\|\mathbf{y}\|_{\mathbf{W}}$ represents the weighted ℓ^2-norm of $\mathbf{y}$.

The (symmetric) correlation matrix can then be computed as $\mathbf{R} = [\rho(i,j)]_{i,j \in \mathcal{A}}$. To identify paradigms, we use the correlation matrix $\mathbf{R}$ and a hierarchical clustering algorithm. For this purpose, we need to compute a dissimilarity (or distance) matrix from $\mathbf{R}$. Several methods are possible; for example, the distance can be defined as follows:

$$d(i,j) = 1 - |\rho(i,j)|, \tag{15}$$

such that negative correlation values are also considered. The paradigmatic algorithm can evaluate more than a single distance function.

We resort to hierarchical clustering [28] because it is superior to partitional clustering for non-isotropic clusters. In fact, the size of the clusters is not fixed beforehand. Moreover, the number of clusters is also unknown; thus, the clusters must be evaluated based on their intragroup distance properties. Our algorithm should also permit the formation of overlapping clusters by setting different thresholds for the average intragroup distance. The pseudocode for the hierarchical clustering algorithm is presented in Algorithm 1.

```
input  : a set of symbols $\mathcal{S}$
output: a hierarchy of clusters with overlaps
Allocate each symbol $s \in \mathcal{S}$ to a single cluster. Let $\mathcal{C}$ be the set of clusters;
Let $\Delta$ denote the diagonal in the Cartesian product $\mathcal{S} \times \mathcal{S}$. $\mathcal{P} = \Delta$;
while $\mathcal{S} \notin \mathcal{C}$ do
    $(a, b) \longleftarrow \mathrm{argmin}_{(\mathcal{S}\times\mathcal{S})\setminus\mathcal{P}}\ d$;
    $\mathcal{P} \longleftarrow \mathcal{P} \cup \{a \times b, b \times a\}$;
    $X \longleftarrow \{a, b\}$;
    forall symbols $c \in \mathcal{S}$ do
        if $d_{average}(X, c) < d(a, b)$ then
            $X \longleftarrow X \cup c$;
        end
    end
    $\mathcal{C} \longleftarrow \mathcal{C} \cup X$;
end
```

Algorithm 1: Hierarchical clustering of symbols that returns partially overlapping clusters.

The paradigmatic algorithm can be run on any set of symbols, including existing syntagms and paradigms, in the iterations following the first one. The syntagmatic and paradigmatic algorithms interact with each other to generate new meaning representations, including syntagms of paradigms and paradigms of paradigms, as depicted in Figure 3. To trigger the first iteration, the semiotic cognitive automaton is required only to be able to identify in the corpus the different base symbols making up the input alphabet $\mathcal{A}$.

3 Learning the semantics of a notational system

In this section, we will demonstrate how the semiotic cognitive automaton can learn, without any a priori representation, the semantics of a notational system, namely, arithmetic operations. From a semiotic perspective, a sequence of concrete examples of arithmetic operations exhibits sufficient redundancy to ensure that rules can be inferred and that learning their general meaning is possible [29]. This typically occurs in the learning of mathematics by humans. The automaton employs the redundancy and regularities of exemplary arithmetic operations to understand their semantics. The only requirement for the automaton is that it must be able to discern the set of base symbols.

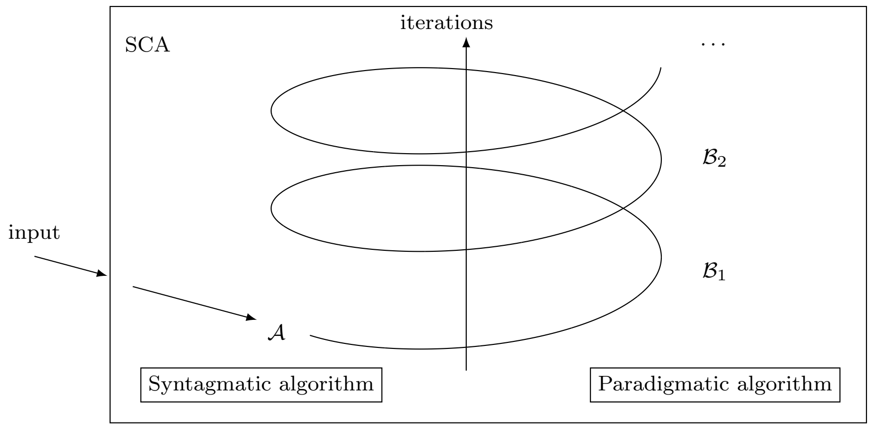

Fig. 3 The semiotic cognitive automaton (SCA) and its main components. Several special sets of symbols are also represented: the input alphabet, $\mathcal{A}$; the set of paradigms created by the first iteration of the paradigmatic algorithm, $\mathcal{B}_1$; and the set of paradigms created by the second iteration of the paradigmatic algorithm, $\mathcal{B}_2$. The syntagmatic and paradigmatic algorithms can use any of the symbols existing in any of the previous iterations.

The semantics of the numerical notational system involves numbers and computations. Numerals are marks that have been more or less arbitrarily chosen to represent particular numerosities. Ciphers have a conventional, symbolic meaning [30], e.g., the Arabic digit '`3`' denotes a particular natural number, namely, 3. In multi-cipher numerals, the basic symbols are concatenated in accordance with certain syntax rules that determine their meaning [31]. In an additive numerical system, concatenation takes the function of addition (as in the Roman numeral '`III`'), whereas a positional system is characterised by a more complex place-value representation. The primary purpose of numerals is to denote numerosities. Moreover, they assist in performing arithmetic operations, such as addition, subtraction and multiplication, wherein, through operational signs, certain syntactical structures of symbols acquire semantic meanings. Arithmetic is associated with hard constraints, i.e., a relation of necessity exists between the expanded form of an operation and its result, represented by the equal sign. Furthermore, calculation can be reduced to simple syntactical symbol manipulation [18]. Therefore, semantics must be addressed explicitly.

Let the corpus be composed of a set of 500 two-operand additions, 500 two-operand subtractions and 500 two-operand multiplications, in decimal notation. The operands are randomly distributed on the interval $[0, 500)$, and only subtractions in $\mathbb{N}_0$ are included. Let the symbol '`;`' separate the different operations. The use of a separator symbol is necessary given our assumption that the input can be regarded as a corpus.

Our alphabet is then defined as $\mathcal{A} = \{\texttt{0}, \texttt{1}, \cdots, \texttt{9}, \texttt{+}, \texttt{-}, \texttt{x}, \texttt{=}, \texttt{;}\}$. Note that the symbol `x` represents the multiplication sign and that the sign '`+`', which looks like a Perl special character, must be escaped in Perl notation, i.e., $\boxed{\texttt{\textbackslash +}}$. Throughout this section, the use of Perl notation is indicated by a bounding box.

3.1 First iteration

In the first iteration, the syntagmatic algorithm is concerned only with adjacency relations of the base symbols, of the form $f(i,j) = \boxed{ij}$, with $i, j \in \mathcal{A}$.

The correlation matrix is computed using Equation 13 and by taking the following vector of differences:

$$\boldsymbol{\delta}_i = \begin{bmatrix} p(i, a_1) \\ \vdots \\ p(i, a_L) \\ p(a_1, i) \\ \vdots \\ p(a_L, i) \end{bmatrix}, \tag{16}$$

in which we assume that the a priori probability of occurrence p_0 is zero for all relations, as is the case before the probabilities of occurrence of the base symbols are computed. Such a choice of $\boldsymbol{\delta}_i$ is necessary because the corpus contains operands that are randomly distributed on the interval $[0, 500)$, and hence, the probabilities of occurrence of the base symbols are misleading.

We visualise the correlation matrix by means of a bidimensional plot in which each cell of the matrix is assigned a colour. Figure 4 shows such a map. It follows from Equation 13 that every value on the diagonal of the map must be 1. Moreover, given our choice of $\boldsymbol{\delta}_i$, the correlations can only be positive. We use these correlations to construct paradigms.

From the correlation matrix, the dissimilarity metric is computed using the method defined in Equation 15. The hierarchical clustering algorithm performs progressive merging of the symbols. We observe that the clusters of ciphers from '`1`' to '`4`' form as a result of the choice of the parameters for the generation of random numbers in the corpus (see the introduction to this section): after a sign '`;`', '`+`', or '`x`', the operand is more likely to begin with one of these ciphers than any other. The subtraction sign, '`-`', exhibits some dissimilarity with the other operational signs, as we require subtractions to remain within $\mathbb{N}_0$.

To demonstrate how hierarchical clustering operates, a type of graph called a dendrogram is typically employed. This is a tree diagram in which leaf nodes are the original elements and the remaining nodes represent the clusters to which they belong. However, a tree diagram cannot represent the partially overlapping clusters obtained from our Algorithm 1. For this reason, we invented a new type of diagram, for which we coined the term *desmogram*, from the Greek word for "link". In a desmogram, a cluster characterised by an average intragroup distance below a given threshold is represented by a link

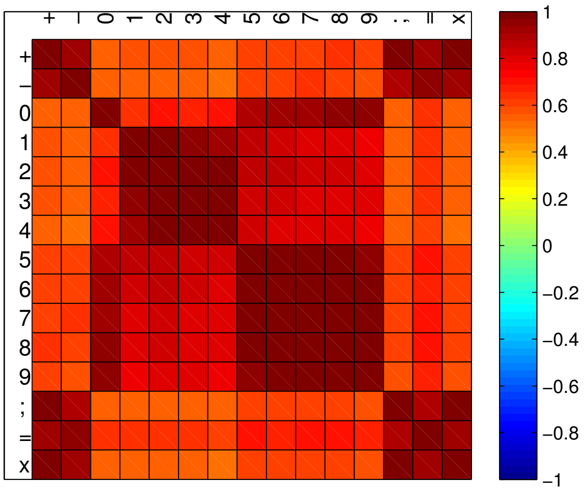

Fig. 4 Correlation map (heatmap) of the base symbols. The symbols are ordered in accordance with their ASCII codes.

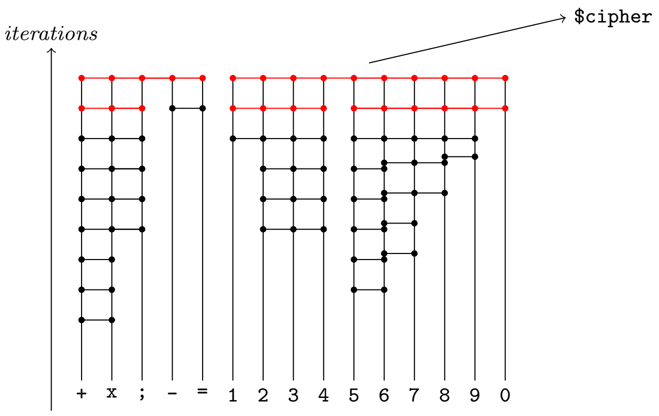

Fig. 5 Desmogram of the first iteration of the automaton. The clusters are ordered from the bottom to the top of the figure in the order of their formation. The clusters with the highest intergroup dissimilarity are formed last. Partially overlapping clusters are possible. Only the clusters highlighted in red are selected based on their average intragroup distances and overlap relations.

connecting multiple points, where each point represents an element and is duplicated in the representation as many times as necessary. From bottom to top, the average distance threshold increases. The desmogram based on the correlation matrix of Figure 4 is shown in Figure 5. To reduce the search space for subsequent syntagmatic analysis, our clustering algorithm also performs cluster selection based on the average intragroup distances and overlap relations of the clusters. The clusters selected from the correlation matrix of Figure 4 are highlighted in red in Figure 5.

Among the selected clusters, we observe the paradigm of the ciphers and the paradigm of the operational signs. For the remainder of this work, we will refer to these clusters as variables, namely, $\boxed{\texttt{\$cipher=[0:9]}}$ and $\boxed{\texttt{\$sign=(\textbackslash+|-|;|=|x)}}$, respectively. (In the following, meaningful names are assigned to these variables solely to facilitate the reader's understanding. The automaton instead refers to them using two nested counters, the first of which identifies the iteration and the second of which identifies the position of the paradigm within the iteration.) At this stage, the automaton considers all selected clusters of symbols, i.e., paradigms, as shown in Figure 5, to have the same importance. Let us refer to the set of clusters created by the automaton in its first iteration as $\mathcal{B}_1$. Table 2 summarises the findings of the first iteration.

Table 2 Output of the first iteration.

Syntagms		Paradigms
`;0`,`;=`, ...	impossible syntagms	$\mathcal{B}_1 \supseteq \{\texttt{\$cipher}, \texttt{\$sign}\}$

3.2 Second iteration

In the second iteration, the syntagmatic algorithm looks for relations that are more complex than simple adjacency. We propose to interpret the syntagmatic algorithm as a process of abduction, deduction and induction. It is commonly accepted that abduction is initiated by a puzzling observation for which an explanation will provide some kind of reward [32]. The first iteration of the automaton reveals the existence of certain paradigms. This is surprising, as such an outcome is not guaranteed by the structure of the paradigmatic algorithm alone. An hypothesis is then formulated that the paradigms from the first iteration enable more complex syntagmatic relationships. As the automaton lacks a deeper understanding from previous iterations, it proceeds via brute force. Several hypotheses are formed at this stage, involving combinations of symbols and paradigms in the sets $\mathcal{A}$ and $\mathcal{B}_1$. Equivalently, we can refer to these hypotheses as regular expressions. These hypotheses must then be tested. Through deduction, the a priori probability $p_0(f)$ of a regular expression f can be computed starting from the occurrence probabilities of its components. Through induction, the relevance of the regular expression f can be computed using Equation 7. An hypothesis fails the test if its relevance falls below the threshold Th.

Table 3 lists the hypotheses that score the maximum relevance and are thus accepted.

Table 3 The first syntagmatic relations involving the paradigm `$cipher`. The numbers of occurrences and the relevance values are also shown.

Regular expression f	$\#(f)$	$r(f)$
`=$cipher+;`	1500	1
`;$cipher+\+`	500	1
`;$cipher+-`	500	1
`;$cipher+x`	500	1
`x$cipher+=`	500	1
`\+$cipher+=`	500	1
`-$cipher+=`	500	1

Regular expressions associated with accepted hypotheses can be further concatenated, and the a priori probabilities of the resulting concatenations can be computed using Equation 2. Two steps of concatenation are possible using the syntagms listed in Table 3, and the results are listed in Table 4. We note that these hypotheses have ill-defined relevances because in each case, the a priori probability of occurrence coincides with the maximum value of the probability of occurrence. These relations do not add any information compared with the relations of Table 3 but rather organise this information in a more structured way. Nevertheless, these hypotheses are accepted given their high number of occurrences.

Table 4 Syntagmatic relations obtained through concatenation. Some of the relations are accepted because of the criteria used to build the corpus. Because $p_{max}(f) = p_0(f)$, the relevance is ill-defined.

Regular expression f	$\#(f)$	$r(f)$
`;$cipher+\+$cipher+=`	500	i.-d.
`;$cipher+-$cipher+=`	500	i.-d.
`;$cipher+x$cipher+=`	500	i.-d.
`=$cipher+;$cipher+\+`	500	i.-d.
`=$cipher+;$cipher+-`	500	i.-d.
`=$cipher+;$cipher+x`	500	i.-d.
`\+$cipher+=$cipher+;`	500	i.-d.
`-$cipher+=$cipher+;`	500	i.-d.
`x$cipher+=$cipher+;`	500	i.-d.
`;$cipher+\+$cipher+=$cipher+;`	500	i.-d.
`;$cipher+-$cipher+=$cipher+;`	500	i.-d.
`;$cipher+x$cipher+=$cipher+;`	500	i.-d.
`=$cipher+;$cipher+x$cipher+=`	500	i.-d.
`=$cipher+;$cipher+-$cipher+=`	500	i.-d.
`=$cipher+;$cipher+\+$cipher+=`	499	i.-d.
`-$cipher+=$cipher+;$cipher+-`	499	i.-d.
`x$cipher+=$cipher+;$cipher+x`	499	i.-d.
`\+$cipher+=$cipher+;$cipher+\+`	498	i.-d.

It is possible to use these relations to construct arguments "from part to whole", i.e., rules of induction of the following type: "*after a '*`+`*' sign, the* *first-next symbol of the paradigm* $\boxed{\texttt{\$sign}}$ *is '*`=`*' with probability 1 and the second-next one is '*`;`*' with probability 1*". Obviously, such rules are valid only in our corpus, given the criteria we followed to assemble it, and are not generally valid in the numerical notational system. In other words, the principle of induction can produce only explanations and not truths [33]. The inductive bias [34] of our algorithm is not so strong, when compared to other machine learning algorithms and thanks to the flexibility guaranteed by the paradigmatic algorithm. For it, it is straightforward, even after only one occurrence of a mathematical sentence with three operands, to hypothesise that $\boxed{\texttt{\$cipher+}}$ and $\boxed{\texttt{\$cipher+\textbackslash+\$cipher+}}$ form a paradigm together and to update the rules of induction.

The rules of induction are not purely syntagmatic because they are found using previously discovered paradigms. Without knowing the paradigms of the first iteration, the search space to retrieve these rules of induction would have been prohibitively large. In the second iteration, only some of the paradigms discovered in the first iteration, i.e., $\boxed{\texttt{\$sign}}$ and $\boxed{\texttt{\$cipher}}$ (see Figure 5) are retained in accepted hypothesis by the syntagmatic algorithm. Nothing, however, prevents paradigms that are not retained in one iteration to be used in subsequent iterations, and therefore, these paradigms are held in memory rather than being deleted.

We use the term *semiotic cognitive grounding* (see the Introduction) to refer to the process that brings meaning to certain paradigms in connection with syntagms expressing rules of induction. Note that a symbol may be used in a context different than that of its creation: the paradigm $\boxed{\texttt{\$cipher}}$ is created in the first iteration on the basis of adjacency relations between symbols and is then used in the second iteration to represent operands separated by operational signs. However, the paradigm is still considered in a first-order statement based solely on analysis of the input. Second-order reasoning about the paradigm `$cipher`, which relies on the ability of the automaton to observe its own structure, is described in the discussion of the third iteration.

At this point in the analysis, no new paradigms will emerge. The syntagmatic algorithm needs to proceed further. An additional method of forming new hypotheses is through *selection*, i.e., choosing one of the members of a paradigm to represent the paradigm. To formulate such an hypothesis, we begin with a syntagmatic relation that makes use of paradigms and suppose that the occurrence of a specific paradigm member can determine a new syntagmatic relation. When faced with a sequence of paradigm members, as in the regular expression $\boxed{\texttt{/\$cipher+/}}$, the hypothesis must specify which position the selected member occupies.

Hypotheses specifying the value taken by $\boxed{\texttt{\$cipher}}$ at only one position and starting from the regular expressions listed in Table 4 fail to produce new syntagmatic relationships, as all instances have approximately null relevance, with one exception[2]. The same can be said for hypotheses specifying the values at two positions. However, if three positions to be specified are chosen in an opportune manner, then only a small subset of the possible instances will score highly in relevance, and the majority of instances will have a null probability of occurrence.

Table 5 shows several instances of such regular expressions, ordered first by relevance and then by frequency. By writing $\boxed{\texttt{\$cipher\{n\}}}$, we indicate that exactly n (unspecified) ciphers match at different positions within the syntagm. Note that only instances that contain the symbol '`+`' have been included in Table 5. In the following, we separate the relations output by the syntagmatic algorithm based on the operational signs they contain. Such a separation can be implemented by the paradigmatic algorithm using different weight vectors, as in Equation 14.

Table 5 Examples of syntagmatic relations obtained through the selection of syntagms containing the addition sign.

Regular expression f	$r(f)$	$\#(f)$	$\#_{max}(f)$
`;4$cipher*\+2$cipher*=7$cipher*;`	1	15	15
`;3$cipher*\+4$cipher*=8$cipher*;`	1	14	14
`;$cipher*0$cipher{n}\+$cipher*4 $cipher{n}=$cipher*4$cipher{n};`	1	11	11
`;$cipher*9$cipher{n}\+$cipher*7 $cipher{n}=$cipher*6$cipher{n};`	1	11	11
`;$cipher*0$cipher{n}\+$cipher*8 $cipher{n}=$cipher*8$cipher{n};`	1	10	10
`;$cipher*7\+$cipher*8=$cipher*5;`	1	9	9
...			
`;$cipher*7$cipher{n}\+$cipher*8 $cipher{n}=$cipher*5$cipher{n};`	0.928	13	14
`;$cipher*2$cipher{n}\+$cipher*5 $cipher{n}=$cipher*7$cipher{n};`	0.922	12	13
`;4$cipher*\+3$cipher*=8$cipher*;`	0.919	12	13
...			
`;$cipher*9$cipher{n}\+$cipher*1 $cipher{n}=$cipher*0$cipher{n};`	0.8	8	10
...			
`\+$cipher*3=$cipher*6;$cipher*3\+`	0.797	4	5
...			
`;$cipher*9$cipher{n}\+$cipher*1 $cipher{n}=$cipher*1$cipher{n};`	0.568	4	7
...			

In the second iteration, the syntagmatic algorithm discovers a structure that is of fundamental importance for understanding the rules governing the positional system (read from right to left). A convenient means of expressing the relations is to represent regular expressions using three variables, which

[2] This exception is represented by the regular expressions that contain the impossible sequences "`;0`", "`=0`",..., already known from the first iteration (see also Table 2).

take their values from among the symbols of the corpus alphabet, in the form $f(i,j,k)$, with $i,j,k \in \mathcal{A}$. A recurrent example from Table 5 is

$$f_{addition}(i,j,k) = \boxed{\texttt{;\$cipher*}}$$

$$\diamond i \diamond \boxed{\texttt{\$cipher\{n\}\textbackslash+\$cipher*}} \diamond j \diamond \boxed{\texttt{\$cipher\{n\} =\$cipher*}} \diamond k \diamond \boxed{\texttt{\$cipher\{n\};}}, \quad (17)$$

but other relations occur as well.

The second iteration of the paradigmatic algorithm makes use of ternary relations, as opposed to the binary relations of adjacency used in the first iteration. The outputs of the second iteration of the paradigmatic algorithm are again paradigms of the elements of $\mathcal{A}$, specifically paradigms of the ciphers $\texttt{0}, \cdots, \texttt{9}$, which are used to specify syntagms through selection by the syntagmatic algorithm. To compute a 10×10 cipher correlation matrix, the values of two ciphers in these ternary relations must be fixed. The correlations can be computed using a vector of relevances, i.e., a generalisation of the vector of differences (see the description of "The paradigmatic algorithm" in Section 2.2). It is not necessary to compute the complete vector of relevances, which would consider all combinations of i, j and k for every ternary relation. To reduce the computational burden of the paradigmatic algorithm, we fix an acceptance threshold of $Th = 0.45$ for the acceptance of syntagms and, for convenience, approximate the relevances of nonexistent or unacceptable syntagms as zero. In place of the exact vector of relevances, we then obtain a sparse vector:

$$Sp\boldsymbol{\delta}_i = \begin{bmatrix} r(f_{addition}(i, \texttt{0}, \cdots, \texttt{9}, \texttt{0}, \cdots, \texttt{9})) \\ r(f_{addition}(\texttt{0}, \cdots, \texttt{9}, i, \texttt{0}, \cdots, \texttt{9})) \\ r(f_{addition}(\texttt{0}, \cdots, \texttt{9}, \texttt{0}, \cdots, \texttt{9}, i)) \\ \vdots \end{bmatrix}, \quad (18)$$

which will include, in addition to the 300 $f_{addition}$ relations, any other relation from Table 5 that has been accepted.

We compute the correlation matrix via sparse vector multiplication. Entry $\rho(i,j)$ is computed as follows:

$$\rho(i,j) = \frac{Sp\boldsymbol{\delta}_i^T Sp\boldsymbol{\delta}_j}{\|Sp\boldsymbol{\delta}_i\| \|Sp\boldsymbol{\delta}_j\|}. \quad (19)$$

The (unweighted) correlation map is shown in Figure 6. The correlations are always non-negative because we required the relevance to be non-negative.

The set of paradigms created during the second iteration $\mathcal{B}_2$ includes the following: $\boxed{\texttt{\$zeroOrOne=(0|1)}}$, $\boxed{\texttt{\$oneOrTwo=(1|2)}}$, $\boxed{\texttt{\$twoOrThree=(2|3)}}$, and so on. At the end of this iteration, more detailed rules of induction can be formed. For example, we can make the following statement: "*after a match to the regular expression* $\boxed{\texttt{;\$cipher*1\$cipher\{n\}\textbackslash+\$cipher*1\$cipher\{n\}=}}$, *the regular expression* $\boxed{\texttt{\$cipher*\$twoOrThree\$cipher\{n\};}}$ *will be found*

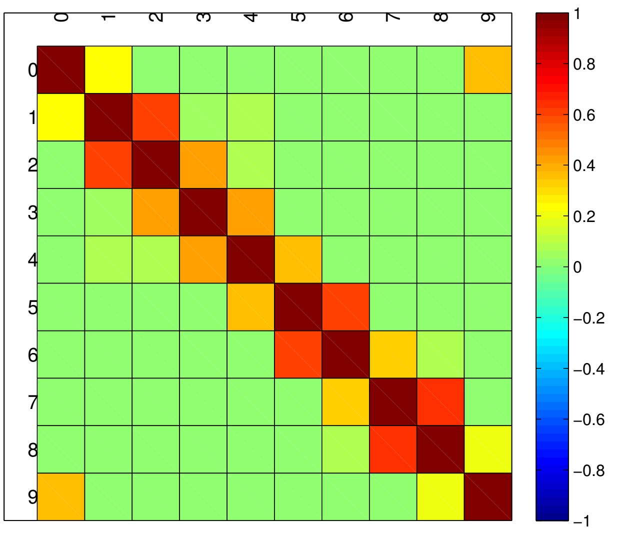

Fig. 6 Second iteration of the paradigmatic algorithm: correlation map for the ciphers obtained using only relations containing the addition sign.

with probability 1". These rules of induction remind us of the facts related to addition, but the automaton cannot understand semantic relationships at the present stage. The second iteration thus reveals the paradigm of consecutive ciphers, i.e., ciphers whose difference is either 1 or, in the case of nine and zero, the base of the numerical system decremented by 1.

Two additional actions can be performed by the automaton at this stage. First, upon observing that all of the new paradigms are pairs, it can group these paradigms into a single one:

```
$adjacentCiphers=(01|10|12|21|23|32|34|43|45|54|56|65|67|76|78
|87|89|98|90|09)
```
.

A paradigm of paradigms is very versatile. In Perl syntax, it is described as a capture buffer [23]. Let us consider the following example: to match the occurrence of consecutive ciphers in a particular regular expression, two capture buffers are created using bracketing constructs, `(···)`: `;$cipher*($cipher)$cipher{n}-$cipher*($cipher)$cipher{n}=`; these buffers are referenced as `$1` and `$2`. There is a match only if the following match operation returns true: `"$1$2"= /$adjacentCiphers/`.

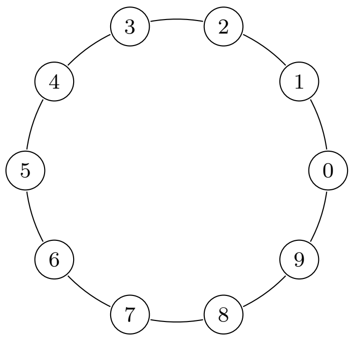

Fig. 7 Graph representation of the paradigms of the second iteration. Two symbols are connected if they appear in the same paradigm.

Next, a convenient representation of the paradigms as a graph is generated. Figure 7 depicts the ciphers as the nodes of a graph. An edge exists between any two ciphers if they appear together in a paradigm. The resulting graph is clearly a chain. This graph was not obtained through construction but rather represents a surprising fact, for which an explanation can be sought through abduction in the subsequent iteration [32]. This representation as a graph suggests a method of building new, tentative paradigms for input to the syntagmatic algorithm without requiring the paradigmatic algorithm to compute the correlation-based distances between symbols. From this graph, we can conclude that 3, for example, has two single-hop neighbours, namely, 2 and 4; two two-hops neighbours, namely, 1 and 5; and so on. Moreover, the following paradigm of pairs is created: $\boxed{\texttt{\$adjacentCiphersAscending=(01|12|23|34|45|56|67|78|89|90)}}$, in which the ciphers appear in only one possible order, consistent with the chain graph.

It rests with the automaton to perform a paradigmatic analysis of the syntagms that contain either one subtraction sign or one multiplication sign.

Let us compute the correlation matrix using only syntagms that contain the symbol '`-`' and have a relevance greater than the threshold. We can set certain weights to zero in **w** of Eq. 14 to isolate the sparse vector:

$$Sp\boldsymbol{\delta}_i = \begin{bmatrix} r(f_{subtraction}(i, \mathtt{0}, \cdots, \mathtt{9}, \mathtt{0}, \cdots, \mathtt{9})) \\ r(f_{subtraction}(\mathtt{0}, \cdots, \mathtt{9}, i, \mathtt{0}, \cdots, \mathtt{9})) \\ r(f_{subtraction}(\mathtt{0}, \cdots, \mathtt{9}, \mathtt{0}, \cdots, \mathtt{9}, i)) \\ \vdots \end{bmatrix}, \mathbf{w} = \begin{bmatrix} \mathbf{1}_{100} \\ \mathbf{1}_{100} \\ \mathbf{1}_{100} \\ \mathbf{0} \end{bmatrix}, \qquad (20)$$

where

$$f_{subtraction}(i, j, k) = \boxed{\texttt{ ;\$cipher*}}$$
$$\diamond i \diamond \boxed{\texttt{\$cipher\{n\}-\$cipher*}} \diamond j \diamond \boxed{\texttt{\$cipher\{n\} =\$cipher*}} \diamond k \diamond \boxed{\texttt{\$cipher\{n\};}}, \qquad (21)$$

($f_{subtraction}$ was constructed in analogy with $f_{addition}$).

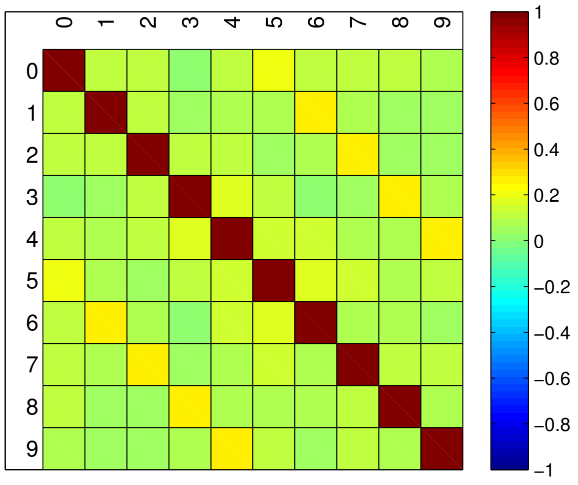

Fig. 8 Second iteration of the paradigmatic algorithm: correlation map for the ciphers obtained using only relations containing the multiplication sign.

The clustering algorithm then returns already established paradigms: $\boxed{\texttt{\$zeroOrOne=(0|1)}}$, $\boxed{\texttt{\$oneOrTwo=(1|2)}}$, ... If relations different from $f_{subtraction}$ are not weighted out, then the correlation matrix contains too much noise because of false rules introduced by the specific criteria used to generate the corpus.

Let us compute the correlation matrix using only syntagms that contain the symbol '`x`' and have a relevance greater than the threshold. The (unweighted) correlation map is shown in Figure 8; the paradigms $\boxed{\texttt{\$zeroOrFive=(0|5)}}$, $\boxed{\texttt{\$oneOrSix=(1|6)}}$, ... are returned, i.e., the paradigms of node pairs of the graph in Figure 7: $\boxed{\texttt{\$fiveHopsCiphers=(05|16|27|38|49|50|61|72|83|94)}}$.

To recapitulate, the second iteration yields the following:

- rules of induction that include the forms of arithmetic two-operand operations;
- a semiotic cognitive grounding of the paradigms `$sign` and `$cipher`; and
- a univocal ordering of the elements in `$cipher`, the order of crescent ciphers, at a stage before it is known that they represent ciphers.

Table 6 summarises the findings of the second iteration.

3.3 Third iteration

As discussed in Section 2.2, the syntagmatic algorithm hypothesises that the new paradigms can form new syntagms, which are associated with a metric

Table 6 Output of the second iteration.

Syntagms
`;$cipher*1$cipher{n}\+$cipher*1$cipher{n}=$cipher*(2\|3)$cipher{n};`
...
`;$cipher*1$cipher{n}-$cipher*9$cipher{n}=$cipher*(1\|2)$cipher{n};`
...
`;$cipher+0x$cipher+=$cipher+0;`
`;$cipher+x$cipher+0=$cipher+0;`
`;$cipher*1x$cipher*($cipher)=$cipher*($cipher); "$1"="$2"`
`;$cipher*($cipher)x$cipher*1=$cipher*($cipher); "$1"="$2"`
Paradigms
paradigms of paradigms of $\mathcal{B}_1$
`$adjacentCiphers`, `$adjacentCiphersAscending`, `$fiveHopsCiphers`

of relevance to determine whether they satisfy the acceptance criterion. Using capture buffers, accepted hypotheses take the following form:

```
;$cipher*($cipher)$cipher{n}\+$cipher*$zeroOrOne$cipher{n}=
                              $cipher*($cipher)$cipher{n};
"$1$2"= /$adjacentCiphersAscending/
```

and

```
;$cipher*($cipher)$cipher{n}-$cipher*($cipher)$cipher{n}=
                              $cipher*$zeroOrOne$cipher{n};
"$2$1"= /$adjacentCiphersAscending/
```

.

In particular, for the last ciphers of the operands and the result, the syntagm is more precise:

```
;$cipher*($cipher)\+$cipher*1=$cipher*($cipher);
"$1$2"= /$adjacentCiphersAscending/
```

and

```
;$cipher*($cipher)-$cipher*($cipher)=$cipher*1;
"$2$1"= /$adjacentCiphersAscending/
```

.

Consider now the additional paradigm of pairs `$twoHopsCiphersAscending=(02|13|24|35|46|57|68|79|80|91)`, derived from the relations of Figure 7. Let us indicate the property of being two hops distant by highlighting in red the descriptive text in the variable name. Thus far, all variable names have been conventional. However, this variable has been deliberately constructed based on this property. The following hypothesis is accepted:

```
;$cipher*($cipher)\+$cipher*2=$cipher*($cipher);
"$1$2"= /$twoHopsCiphersAscending/
```

.

Other sets can be constructed similarly based on the graph of Figure 7. Obviously, `$adjacentCiphersAscending` is the set of ciphers that are one hop distant. A puzzling observation is made by the automaton: the hop distance of the symbols in terms of their paradigmatic relations appears to be related to certain symbols, e.g., adjacent symbols are related to symbol '1' and two-hop-distant symbols to symbol '2', and furthermore, every time the hop distance is increased by one unit, the symbol related to the increased distance happens to be adjacent to the symbol related to the initial distance in Figure 7; '`1`' and '`2`' are such symbols in the example. No counting ability is required of the automaton other than mastering the operation of succession (i.e., iterated counting), which is necessary to move from one node to the next, always in the same direction. Abduction then connects the concept of succession with this observation to formulate the hypothesis that the symbols in the paradigm $\boxed{\texttt{\$cipher}}$ indicate numerosities.

This hypothesis is confirmed for all ciphers. In particular, '`0`' can be associated with a numerosity of 10. However, ten-hop-distant ciphers are simply identical ciphers, suggesting that '`0`' is a neutral element. Similarly, there is no need to attach ambiguity to any other cipher, e.g., by associating '`1`' with not only a numerosity of 1 but also numerosities of 11, 21, and so on; our automaton implements Occam's razor. The automaton knows how to represent only the numbers up to nine in the notational system.

There is no way to represent this knowledge of the automaton by means of regular expressions. This knowledge, however, amounts to *semiotic cognitive grounding* of the *second order* (see the Introduction).

There is no doubt that initially, the ciphers are merely input symbols with no attached meaning, to which the paradigmatic algorithm and iterative semiotic modelling can be applied. The automaton then, by computing the relevancy of hypotheses associated with regular expressions, learns that the paradigm corresponding to ciphers provides explanatory rules about the corpus, i.e., rules of induction. The paradigm of ciphers, which results from a *semiotic* definition, is inherently and meaningfully *grounded* in the *cognitive* process directed at a corpus of mathematical sentences. The first-order meaning of a semiotic symbol is defined only in relation to rules of induction in the corpus. However, a second-order description of the symbol is also possible. The automaton constructs an internal representation of paradigm elements, i.e., the representation of the ciphers shown in Figure 7, and forms a connection between this representation and the structures learned from the corpus to assign them their semantic meaning of numbers. Second-order semiotic grounding requires a high level of abductive reasoning ability. The automaton could continue iterating between the syntagmatic and paradigmatic algorithms, still arriving (as will be shown in the following) at the capability of predicting the results of arithmetic operations with no understanding of their semantic meaning. Our semiotic algorithm, however, must be capable of second-order reasoning about symbols, in a manner unprecedented in the development of artificial intelligence.

The problem is then encountered that it is uncommon to have a program that can "master the operation of succession" and "perform abduction". The question of which instructions would endow the program with the necessary intentionality to intrinsically ground symbols remains open. It is, however, both possible and relatively simple to instruct the automaton to make observations of its internal storage and to relate these observations to the input symbols in all combinatorially possible meaningful ways. The abduction process then enables the selection of certain results from among these tentative relations. Our proposal is the following.

The semiotic cognitive automaton has access to a function f, which takes as an input an item list, $list[]$; the identifier of an item in the list, $init$; and an offset value, p, which can be zero, one, and so on. The programmer should not restrict p to be an integer type of a particular programming language but instead should use Peano arithmetic numbers, which are constructed iteratively using only one symbol, i.e., the Peano zero 0, and one function, i.e., the Peano successor function $S(\cdot)$, such that $1 = S(0)$, $2 = S(S(0))$, and so on. When $p = 0$, the function returns the initial item $init$; for $p > 0$, the element returns the item that is p hops away from the input item, if such an item exists in the list. Therefore,

$$(item)res = f((item[\cdot])list[\cdot], (item)init, p). \tag{22}$$

The function f for the automaton may be a codelet, written by its programmer, or a routine, hardwired into its circuits. When executing its program, the automaton may call the function f with a certain frequency. For example, suppose that the syntagmatic algorithm is considering a symbol '`+`' appearing in a given position in the corpus and that it is interested in a relation of subsequence. The program could then define $list[]$ as the corpus and $init$ as pointing to the symbol '`+`', set $p = 1$ and call the function f to obtain another symbol as a result. Moreover, the automaton is free to call the function f with the input parameters of its choice. At a certain point, the automaton applies the function f to any of the symbol lists that can be derived from Figure 7, i.e., $list_0 = [\text{'0'}, \text{'1'}, \dots, \text{'9'}]$, $list_1 = [\text{'1'}, \dots, \text{'9'}, \text{'0'}]$ and so on. The results include $f(list_0, \text{'1'}, 0) = \text{'1'}$, $f(list_0, \text{'0'}, 9) = \text{'9'}$, …, $f(list_0, \text{'9'}, 0) = \text{'9'}$; $f(list_1, \text{'1'}, 0) = \text{'1'}$, …, $f(list_1, \text{'9'}, 1) = \text{'0'}$; and so on. The results of running the function f are written into a record of triplets $(init, p, res)$, in which p is a Peano arithmetic number. The automaton compares the entries in this table with the triplets of symbol items in $f_{addition}$. Mappings from a Peano arithmetic number to a digit symbol are then learned: $d(0) = \text{'0'}$, $d(1) = \text{'1'}$, …, $d(9) = \text{'9'}$.

By inverting these relationships, mappings from a digit symbol to a Peano arithmetic number can be formed:

$$p(\text{'0'}) = 0, p(\text{'1'}) = 1, \dots, p(\text{'9'}) = 9. \tag{23}$$

Because the Peano zero is mapped to the symbol '`0`', the only consistent listing is $list_0$. In fact, for any different listing $list_1$ to $list_9$, it holds that $f(list, \text{'9'}, 1) = \text{'0'}$, whereas the successor of $p(\text{'9'})$ cannot be $p(\text{'0'})$. The automaton learns a representation of the numbers up to nine by virtue of the mapping function d, which is defined only for numbers up to nine.

In the following, we will highlight in red the paradigm $\boxed{\texttt{\$cipher}}$ to indicate that its meaning has been learned.

Additionally, the rules of induction require the grounding to become addition facts: “$0 + 0 = 0$”, “$0 + 1 = 1$”, “$1 + 1 = 2$”, and so on. The syntagm `;$cipher*1\+$cipher*1=$cipher*2;` is known from the second iteration of the syntagmatic algorithm. The difference is that in the third iteration, the meanings of the symbols ‘1’, ‘2’ and all other ciphers have been learned by the automaton (the ciphers are then indicated in red in the syntagm). It is then possible, in the third iteration, to interpret the syntagm above in light of the graph of Figure 7. Cipher ‘2’ is one hop from cipher ‘1’. The automaton now interprets said syntagm to be a symbolic representation of the property of “being one hop distant”, i.e., of a unitary increment. This is achieved by distinguishing the three symbols that identify numerosities - i.e., the ciphers ‘1’, ‘1’ and ‘2’ - from the addition sign and the equality sign. Moreover, the addition fact “$1 + 1 = 2$” is obtained as a generalisation of existing syntagms. Such a process is followed for all addition facts for which the result is less than ten. In fact, the automaton does not know how to represent numbers of ten or higher in the notational system.

The place-value representation has not yet been learned. Consequently, only addition facts with results up to nine can be correctly reconstructed. The automaton may incorrectly conclude that “$9+1 = 0$”, which is true only when it is assumed that ‘9’, ‘1’ and ‘0’ represent any of the numerosities $9, 19, 29, \ldots$; any of the numerosities $1, 11, 21, \ldots$; and any of the numerosities $0, 10, 20, \ldots$, respectively (such an explanation, however, is not favoured by Occam’s razor; see above). Let us suppose that the automaton restricts itself only to correct, single-digit addition facts. Still, a number of syntagms that seem to violate the addition facts, i.e., rules of induction in positions other than the right-most one, such as `;$cipher*1$cipher{n}\+$cipher*1$cipher{n}=$cipher*3$cipher{n};`, are known to it. The automaton is left wondering what causes the known addition facts to be disattended at certain positions in the input.

All subtraction facts “$0 - 0 = 0$”, “$1 - 0 = 1$”, “$1 - 1 = 0$”, and so on up to “$9 - 9 = 0$” are also learned. The complementarity of addition and subtraction (additive inverse), which leads to number facts involving the same ciphers, becomes grounded in the construction mechanism using the graph of Figure 7.

By contrast, the automaton cannot yet derive multiplication facts. It could generalise rules of induction for ciphers in the last position on the right-hand side of operands, with the result (see also the syntagms listed in Table 6) of defining zero as the absorbing element of multiplication and one as the multiplicative identity, but it would not be able to ground the operation of multiplication because no rule of induction for positions other than the last one on the right could be formed and too few observations in the corpus support correct single-digit constructs, such as “$2 \times 2 = 4$”, “$2 \times 3 = 6$”, “$2 \times 4 = 8$”, “$3 \times 2 = 6$”, “$3 \times 3 = 9$” and “$4 \times 2 = 8$”.

The third iteration is not yet complete. The syntagmatic algorithm selects hypotheses involving further selection. These hypotheses have the following formats:

```
;$cipher*10$cipher{n}\+$cipher*10$cipher{n}=$cipher*2$cipher{n+1};
```

```
;$cipher*10$cipher{n}\+$cipher*11$cipher{n}=$cipher*2$cipher{n+1};
```

and so on as well as

```
;$cipher*11$cipher{n}\+$cipher*19$cipher{n}=$cipher*3$cipher{n+1};
```

```
;$cipher*12$cipher{n}\+$cipher*18$cipher{n}=$cipher*3$cipher{n+1};
```

and so on.

The paradigmatic algorithm computes the correlations of pairs of ciphers by specifying only one parameter in ternary relations and, through the hierarchical clustering algorithm, admitting the cluster overlaps introduced in Section 2.2. The following paradigms are added to $\mathcal{B}_3$:

– `$ciphersSummingEightOrLess=(00|01|02|···|08|11|12|···|17|22|23|···|26|33|34|35|44)`,
– `$ciphersSummingTenOrMore=(19|28|29|37|···|58|59)`, and
– `$ciphersSummingNine=(09|18|27|36|45)`.

To understand the origin of these paradigms, one should consider, for example, the differences among "51"$\diamond$"x"$\diamond$"+12"$\diamond$"y", "58"$\diamond$"x"$\diamond$"+12"$\diamond$"y" and "57"$\diamond$"x"$\diamond$"+12"$\diamond$"y", where "x" and "y" are placeholders for sets of ciphers of equal length and $\diamond$ denotes string concatenation.

A similar analysis is performed for subtraction. A paradigm of ordered ciphers, consistent with that created by the automaton in the second iteration, is returned in the following form:

```
$ascendingTwoCiphers=(01|02|03|···|09|12|13|···|19|
23|24|···|29|···|78|79|89)
```

.

This reflects, for example, the differences among "58"$\diamond$"x"$\diamond$"-12"$\diamond$"y", "58"$\diamond$"x"$\diamond$"-19"$\diamond$"y" and "58"$\diamond$"x"$\diamond$"-18"$\diamond$"y". Subtraction is not commutative. It is the relationship of inequality in which exist the corresponding ciphers in the subtrahend and the minuend that disambiguates the subtraction rules for ciphers other than the right-most one.

To recapitulate, the third iteration yields the following:

– *semiotic cognitive grounding* of the symbols that represent ciphers,
– synthetic rules of induction for addition and subtraction,
– addition facts up to nine and subtraction facts in $\mathbb{N}_0$, and
– the complimentarity between addition and subtraction.

Table 7 summarises the findings of the third iteration.

Table 7 Output of the third iteration.

Syntagms
`;$cipher*11$cipher{n}\+$cipher*1(0\|1\|···\|7)$cipher{n}=$cipher*2$cipher{n+1};`
`;$cipher*27$cipher{n}-$cipher*1(8\|9)$cipher{n}=$cipher*0$cipher{n+1};`
addition facts up to 9: $0+0=0$, $0+1=1$, ..., $8+1=9$, $9+0=9$
subtraction facts in $\mathbb{N}_0$: $0-0=0$, $1-0=1$, $1-1=0$, ..., $9-9=0$
Paradigms
`$ciphersSummingEightOrLess`,...
`$ascendingTwoCiphers`,...

3.4 Fourth iteration

The semiotic cognitive automaton can correctly predict the result of any operation of addition or subtraction. However, it only knows addition facts with results up to 9 and single-digit subtraction facts. The complexities introduced by place value have been regarded as purely notational. Paradigms from the third iteration, e.g., $\boxed{\texttt{\$ciphersSummingEightOrLess}}$ and $\boxed{\texttt{\$ciphersSummingTenOrMore}}$, are used to create new syntagms, which enable the algorithm to state rules such as *"in a two-operand addition, if ciphers summing to* 2 *or* 12 *appear in the* n*-th positions from the right of the operands and if ciphers summing to* 10 *or more appear in the* $(n-1)$*-th positions from the right of the operands or, when ciphers summing to* 9 *appear in the* $(n-1)$*-th positions from the right of the operands, if ciphers summing to* 10 *or more appearing in the* $(n-2)$*-th positions from the right of the operands, then in the* n*-th position from the right of the result, the cipher '3' appears with probability 1."* By applying typographical rules to strings [18], the automaton can compute that `9+3=12`, but it does not know "$9+3=12$" as an addition fact, as the semantics of place-value representation have thus far escaped it.

The automaton reasons based on the paradigms of the third iteration and on the graph of Figure 7. The synthetic rules of induction for addition show that the cipher in the result is associated with a unitary increment based on the presence on the right of ciphers in $\boxed{\texttt{\$ciphersSummingTenOrMore}}$, i.e., of ciphers that sum to numbers whose symbolic representations are unknown to the automaton (the function d that maps Peano arithmetic numbers to numeral signs, introduced in the previous section, is defined only for $0, \dots, 9$). However, the automaton does know, from syntagms that predict the last cipher on the right in a sum, that these ciphers are associated with a third cipher, which is the one at which one arrives when one starts from the first cipher in the graph of Figure 7 and takes a number of hops given by the other one. These operations have the following two properties: (i) when performing them on the ring, one is certain to step once through the cipher '`0`', and (ii) in the sum, the previous cipher on the left is incremented by one. Through abduction, the automaton learns that the number of passages through zero is the carryover that is added to the previous cipher on the left. In the notation introduced in the previous section (see Eq. 23), the automaton learns that

$$p(\text{“}\mathtt{x}\text{”} \diamond \text{‘}\mathtt{y}\text{’}) = p(\text{‘}\mathtt{y}\text{’}) + 10 \times p(\text{“}\mathtt{x}\text{”}), \qquad (24)$$

where ‘y’ represents a generic cipher, “x” represents any set of ciphers, and $\diamond$ denotes their concatenation. The automaton then computes and stores all remaining single-digit addition facts (the decimal number system contains 100 addition facts [29]), thereby allowing multi-digit addition operations to be solved using this representation, which is more synthetic than the rules based on the third-iteration paradigms $\boxed{\texttt{\$ciphersSummingEightOrLess}}$ and $\boxed{\texttt{\$ciphersSummingTenOrMore}}$.

Note that the automaton already knows from the third iteration that addition and subtraction are inverse operations. Therefore, it can also easily learn how to perform subtraction with borrowing.

Thus, the semiotic cognitive automaton has autonomously understood the meaning of the numerical system, its syntax and its semantics. It not only can solve any operation of addition and subtraction, or of counting forwards and backwards, but also can perform original mathematical reasoning. This occurs as soon as it begins to manipulate symbols independently from the input that it has received. Consider the following two examples.

First, the automaton can determine that the direction of the equality sign is reversible. Numbers in place-value representations are compound numeral signs in which the ciphers, depending on their positions, represent different numerals, all of which are multiples of different powers of 10 and can be combined via addition. Let the automaton consider all possible addition operations that generate a given number (equivalence classes of addition pairs). By recognising the regularity of this set of operations, the automaton discovers the decimal decomposition of a number.

Second, the automaton is not restricted to subtracting only a smaller number from the minuend. Let the automaton consider the subtraction of a larger number from a smaller one. Through the extension of the subtraction facts that it knows, the automaton discovers negative numbers.

It remains only to discuss multiplication. Thus far, the automaton has been unable to generate extensive rules of multiplication through semiotic modelling. Its only rules of multiplication involve ciphers in the last positions on the right. We would like the automaton to receive as an input more relations of the following type: $\boxed{\texttt{;\$cipher*06x\$cipher*6=\$cipher*36;}}$. Alternatively, in an approach analogous to the learning of arithmetic by humans, we provide the automaton with a multiplication table, in parallel with the corpus, to allow multiplication facts to be learned “by rote”. By abduction, the automaton learns that

$$p(d(S(p(\text{‘a’}))) \diamond \text{“}\mathtt{x}\ \text{b”}) = p(\text{“a}\ \mathtt{x}\ \text{b”}) + p(\text{‘b’}), \qquad (25)$$

where ‘a’ and ‘b’ represent any cipher and $S(\cdot)$ is the Peano successor function. Then, the automaton will again consider the multiplications that appear in the

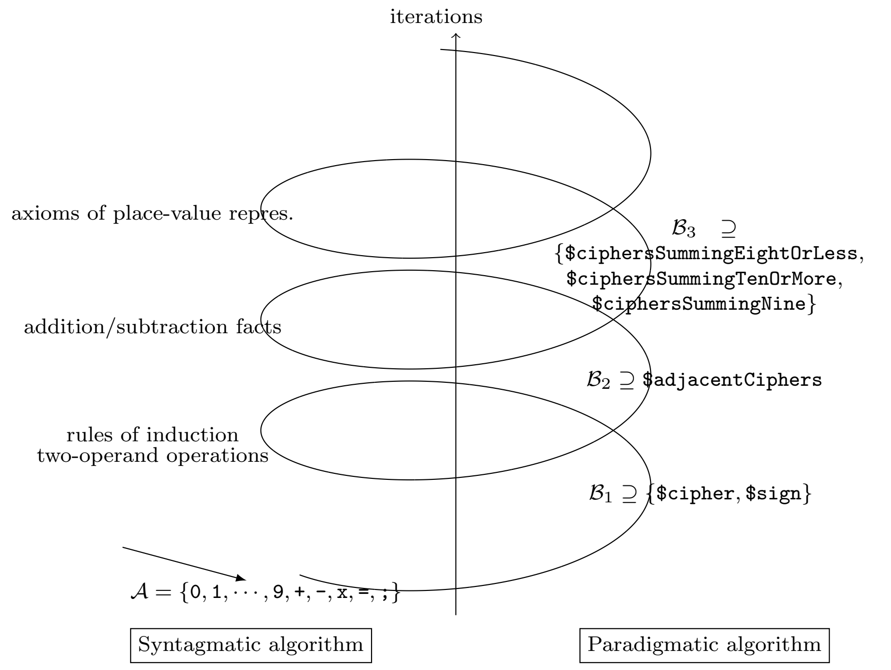
iterations
axioms of place-value repres.
addition/subtraction facts
rules of induction
two-operand operations
$\mathcal{B}_3 \supseteq$ {$cipher$SummingEightOrLess, $ciphersSummingTenOrMore, $ciphersSummingNine}
$\mathcal{B}_2 \supseteq$ $adjacentCiphers
$\mathcal{B}_1 \supseteq$ {$cipher, $sign}
$\mathcal{A} = \{0, 1, \cdots, 9, +, -, x, =, ;\}$
Syntagmatic algorithm
Paradigmatic algorithm

Fig. 9 Paradigms created by the semiotic cognitive automaton upon receiving arithmetic operations as input.

corpus in an attempt to apply this new knowledge. Recall that the automaton has learned the semantics of place-value representation and of the carrying operation. The known multiplication facts can then be combined via the distributive property, e.g., "$51 \times 18 = (5 \times 1) \times 100 + (5 \times 8) \times 10 + (1 \times 1) \times 10 + (1 \times 8)$".

4 Discussion

4.1 The problem of understanding

The semiotic cognitive automaton follows an iterative process through the syntagmatic and paradigmatic algorithms to generate representational formats corresponding to different levels of knowledge and, in 4 iterations, learns how to perform arithmetic operations. The symbolic structures thus generated are intrinsically grounded, as products of semiotic modelling. However, in another sense, we would require a symbol such as '8', for example, to be truly understood as a numerosity before conceding that the automaton grasps semantics. Our proposed algorithm can accomplish this, as well.

To illustrate this, consider the case of an Aboriginal cryptographer who lives in the Australian bush and was never taught what ciphers are, such that initially, they represent for her merely various incomprehensible symbols. She is, however, so gifted that she does not need anybody to tell her that the symbol '8' is the number eight or to teach her arithmetic because she can autonomously discover the arithmetic symbolism by applying the semiotic algorithm to a corpus of mathematical sentences. When applying the proposed algorithm, specifically the third iteration (see Section 3.3), she would be in the following situation: she would know how to use certain syntagms, those containing the addition sign and the cipher '1', as rules of induction, and she would know how to define - via a psychological or tool-assisted operation of succession - a hop distance between any two ciphers using the structures of certain paradigms of cardinality two (for example, ciphers that appear together in a paradigm are said to be one hop distant). She would then connect the operation of succession with the rules derived from the corpus and thereby ground the originally incomprehensible symbol '1' as a representation of the numerosity of one, also doing the same for all other ciphers. Second-order reasoning is easier for such a semiotic cognitive Aborigine to achieve than it is to achieve in a program (although it is not impossible to imagine such a program, as shown in Section 3.3).

Such a program would be able to overcome Searle's Chinese room argument [3]. In this famous thought experiment, previously mentioned in the Introduction, John Searle, who does not speak Chinese, follows the instructions of a Chinese room, i.e., an artificially intelligent chatbot operating in the Chinese language. The Chinese room has been programmed by someone who knows Chinese to have the capability of holding a conversation in Chinese. However, no matter how sophisticated the program may be, regardless of whether it deploys a rule book or an artificial neural network, the program does not exhibit any capability of grounding, and therefore, Searle could not learn Chinese by following the instructions of the program. The situation would be different if he were to receive as an input a corpus of Chinese text and he were to follow some sort of semiotic algorithm - of which no such proposal has yet been described for natural language processing (let alone for non-alphabetic languages) - and to apply second-order reasoning until semantic interpretation were to eventually emerge.

The key here is that the semiotic algorithm teaches how to assemble an enormous constellation of representations, each of them able to appear in a relation with any other such that meaning resides in these relations, and also how to represent within this constellation its own processes and internal states. The requirement established by the Chinese room argument was recently strengthened [35]: could a computer CPU operating on raw binary data really "understand" its program in a manner analogous to the understanding of a human (hence, to the understanding of the Aboriginal cryptographer or to that of Searle in our special Chinese library room)? This question is indeed tricky to answer and relies on the ability for the constellation of representations to generate concrete dynamics of interpretation for the program. The constellation of representations cannot then be reduced to a semantic network, considered just as a directed, labeled graph [36]. We believe that a more productive answer to this question is to analyse the performance of the program in cognitive tests.

It is possible, using a single test, to assess different levels of understanding (as stated in the Introduction and following the suggestion of an anonymous reviewer, Bloom's taxonomy [15] can provide guidance in this task). If a program fails the assigned test, it must be concluded that the program has not yet learned the skills necessary to pass it, i.e., that it has not yet been able to establish certain relations among its constituent entities. The test proposed here differs in spirit from the famous Turing test because its goal is not to fool a judge (e.g., into believing that an artificial intelligence is a human) but rather to assess cognitive capabilities beyond subjectivity. The advantage of creating tests for programs rather than for humans is that it is possible to control all operating conditions of a program.

Our semiotic cognitive automaton operates in the domain of mathematics. Let us provide the automaton with the following test: "`iii+iiiii=`", in which it is confronted with a previously unknown base symbol '`i`'. We know, therefore, that the automaton does not know any syntactical rule to use to answer this test (e.g., a rule that could have been stored as the Perl regular expression `i.{n}\+i.{m}=i.{n+m}`). There exists, however, a set of answers that the automaton could produce that make sense from a syntactical perspective; it could simply hypothesise that the symbol '`i`' forms a paradigm together with the symbol '`1`' and return the answer "`iii222`"; or it could return the answer "`iii444`", predicated on the hypothesis of a paradigm `(i|2)`; and so on. Only if the automaton understands the semantic meaning of numbers can it answer "`iiiiiiii`", or '`8`', or even "`8xi`", each of these answers revealing a higher level of understanding compared with the previous ones.

Consider also the test "`4;7;10;13;`", which cannot be solved directly by syntactical means, as our automaton has received as an input only numeric sentences and not numeric sequences. The automaton then retrieves all of the rules it has learned that contain the numbers 4, 7, 10 and 13, including the addition facts "$4 + 3 = 7$" and "$7 + 3 = 10$". Struck by this parallelism, it generates the hypothesis that an occurrence of a separator '`;`' represents an operation of adding 3 (or, without requiring semantic knowledge, that it represents a synonym for the syntagm "`+3=`"). The hypothesis is accepted upon verifying that "$10 + 3 = 13$". Then, the automaton makes the further hypothesis that after the final separator '`;`', the number 16 should follow. What would happen if we could query the automaton (i.e., ask it and be understood) about the 1000-th number in such a sequence or its generic n-th number? Could we induce the program - for example, by providing it with a relevant textbook - to do algebra (with one or more variables) or to solve word problems?

Moreover, the problem of completing a sequence is reminiscent of an application of Simon Colton's mathematical discovery system HR [37] to number theory. HR takes tables of number facts as input and is programmed with a limited set of production rules to generate sequences of numbers. As demonstrated by Hofstadter [18], a rule such as "take the prime numbers" can be reduced to a mere syntactical operation. No semantics is involved; HR explores

- through an heuristic search - the combinatorial space of all possible rules. Could we induce the program - for example, by providing it with a collection of human-valued mathematical entities, rather than only with worthless arithmetic operations - to formulate its own criteria for relevant mathematical production?

4.2 Relations to other work

The semiotic cognitive automaton described here is unique in its exhibition of second-order reasoning concerning the input under observation. The role of semiotics in automated learning, however, has been noted before. The procedural model of semiotic cognitive information processing is attributed to Burghard Rieger [14]. Such a learning process relies on no pre-established semantics but rather leads to the self-organising discovery, through semiotic modelling, of sign structures from the potential meaning characteristics of a given input and thus enables their interpretation as signifiers of something else. The learnt representations (in the form of syntagms and paradigms) are organised across various levels of meaning and are used to build higher-level representations.

Rieger applied semiotic cognitive information processing to natural language analysis, laying the foundations for *computational semiotics* [27]. In this approach, two steps of abstraction are employed to compute the semantic space of word meanings. In the first step, syntagmatic abstraction, the usage correlations of words in sample tests are computed. In the second step, paradigmatic abstraction, the similarities of distribution over all words are measured to generate semantic spaces. Our implementation of the semiotic cognitive automaton differs from Rieger's implementation in that we perform semiotic modelling as a truly iterative process.

Methods of learning from raw corpora have also been proposed in the field of grammar induction, i.e., the learning of grammatical structures from raw texts [38].

The idea dates back to Harris [39], who postulated that a distributional analysis of partially aligned sequential contexts permits the identification of linguistic units in unannotated text. As early as 1961, Lamb [40] described an algorithm capable of identifying "H-groups", i.e., horizontal groupings of items, including V-groups, that occur sequentially in a syntagmatic construction, and "V-groups", i.e., vertical groupings of items, including H-groups, that occur in similar contexts in a paradigmatic construction. His algorithm performs a statistical analysis of word adjacency. More recently, Solan et al. [41] developed ADIOS (for the automatic distillation of structure), an algorithm for unsupervised grammar induction that is corpus-independent.

ADIOS combines statistical computation and rule learning; it identifies "significant patterns" in the corpus, which is treated as a pseudograph, and forms "equivalence classes", in which strings appear to be interchangeable in a given context. The search for significant patterns is repeated iteratively, and the equivalence classes are replaced in the corpus (graph rewiring) to arrive at further generalisations until no new significant patterns are found. The significant patterns that have been identified are then restated in the form of rewriting rules, yielding the end product of ADIOS in the form of a grammar.

The paradigmatic algorithm of ADIOS returns equivalence classes, i.e., paradigms for which aligned contexts (partially overlapping strings) exist. In contrast to ADIOS, our algorithm can retrieve paradigms for which incomplete information exists by using extended correlation matrices. The syntagmatic algorithm of ADIOS returns syntagms of both base symbols and equivalence classes. To cope with the intractability introduced by an unbounded number of syntagms, ADIOS applies to the graph that represents the corpus a greedy algorithm for selecting the best patterns and, when iterating, considers only syntagms in which such best patterns, i.e., equivalence classes, have been replaced. Our algorithm extends ADIOS, and grammar induction methods in general, through the use of abduction to generate hypotheses.

5 Conclusion

To advance the status of the debate regarding the capabilities of symbolic processing, this paper proposes a semiotic cognitive automaton, a prototype of a multi-stage syntagmatic and paradigmatic algorithm that can successfully discover syntactic and semantic relationships in the case of an "easy" problem of semiotic modelling.

The simplicity of the considered problem - the learning of arithmetic operations - originates from the presence of "hard constraints" through which rules can be inferred by the automaton. In fact, even if the relevant notational system can be easily formalised, the automaton is not provided with this representation beforehand but rather automatically learns it.

Figure 9 summarises the iterations followed by the semiotic cognitive automaton, which receives as an input arithmetic operations in the decimal number system. The paradigms of ciphers and their ordering are discovered via paradigmatic analysis in the early stages of the algorithm; however, they become grounded only in later stages of semiotic modelling. Note that the automaton can learn independently of any specific "language", i.e., independently of a given system of numbers. The automaton can learn arithmetic starting from input operations presented in Roman numerals, operations coded in ASCII and expressed in hexadecimal or in any other format, or even operations expressed in natural language (with complex numerals composed linguistically using multiplication and/or addition [42]); in this last case, the input alphabet would be $\mathcal{A} = \{\texttt{one}, \texttt{two}, \texttt{three}, \ldots, \texttt{ten}, \texttt{eleven}, \texttt{twelve}, \texttt{thir}, \texttt{teen}, \texttt{fif}, \texttt{twen}, \texttt{ty}, \texttt{ty-}, \texttt{for}, \texttt{hundred}, \texttt{and}, \texttt{plus}, \texttt{minus}, \texttt{times}, \texttt{is equal to}\}$ and could be extracted by applying a test for statistically relevant k-strings using a high-order Markov model [43]. The results thus obtained should be compared to those of Figure 9.

Through the application of reasoning to the internal representation of the structures corresponding to ciphers and their ordering, later stages of the algorithm will endow these structures with their authentic meanings through *semiotic cognitive grounding* of the *second order*. Subsequently, automatic and grounded mathematical reasoning becomes possible.

Refinement of the constituent algorithms of our automaton could enable its application to a class of knowledge domains, which could yield additional examples of second-order reasoning.

References

1. A. Newell and H. A. Simon, "Computer science as empirical inquiry: Symbols and search," *Commun. ACM*, vol. 19, pp. 113–126, Mar. 1976.
2. D. Hofstadter, "The ineradicable Eliza effect and its dangers," in *Fluid Concepts and Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought* (D. Hofstadter, ed.), pp. 155–168, Basic Books, 1994.
3. J. Searle, "Minds, brains and programs," *Behavioral and Brain Sciences*, vol. 3, pp. 417–424, 1980.
4. S. Harnad, "The symbol grounding problem," *Physica D: Nonlinear Phenomena*, vol. 42, pp. 335–346, 1990.
5. P. Haikonen, "The role of the associative processing in cognitive computation," *Cognitive Computation*, vol. 1, pp. 42–49, 2009.
6. P. Wang, "Experience-grounded semantics: a theory for intelligent systems," *Cognitive Systems Research*, vol. 6, no. 4, pp. 282–302, 2005.
7. T. Ziemke, "Rethinking grounding," in *Understanding representation in the cognitive sciences* (M. P. A. Riegler and A. von Stein, eds.), (New York), pp. 177–190, Kluwer Academic/Plenum Publishers, 1999.
8. K. J. Devlin, *Introduction to mathematical thinking.* Keith Devlin, 2012.
9. J. H. Fetzer, *Artificial Intelligence: Its Scope and Limits.* Springer Netherlands, 1990.
10. S. A. Jackson and N. E. Sharkey, "Grounding computational engines," in *Integration of Natural Language and Vision Processing* (P. Mc Kevitt, ed.), pp. 167–184, Springer Netherlands, 1996.
11. A. Gomes, R. Gudwin, C. El-Hani, and J. Queiroz, "Towards the emergence of meaning processes in computers from Peircean semiotics," *Mind and Society: Cognitive Studies in Economics and Social Sciences*, vol. 6, pp. 173–187, November 2007.
12. A. Meystel, "Multiresolutional semiotic systems," in *Proc. of the IEEE Int. Symp. on Intelligent Control/Intelligent Systems and Semiotics*, pp. 198–202, 1999.
13. P. Vogt, "The physical symbol grounding problem," *Cognitive Systems Research*, vol. 3, no. 3, pp. 429–457, 2002.
14. B. B. Rieger, "Semiotics and computational linguistics. On semiotic cognitive information processing," in *Computing with words in information/intelligent systems* (L. A. Zadeh and J. Kacprzyk, eds.), (Heidelberg, Germany), pp. 93–118, Physica, 1999.
15. B. Bloom, ed., *Taxonomy of educational objectives: Book I, cognitive domain.* New York: Longman Green, 1956.
16. L. Franco and S. A. Cannas, "Solving arithmetic problems using feed-forward neural networks," *Neurocomputing*, vol. 18, pp. 61–79, 1998.
17. Y. Hoshen and S. Peleg, "Visual learning of arithmetic operations," *CoRR*, vol. abs/1506.02264, 2015.
18. D. Hofstadter, "How Raymond Smullyan inspired my 1112-year-old self," in *Four Lives: a Celebration of Raymond Smullyan* (R. Smullyan and J. Rosenhouse, eds.), Dover Publications, 2014.
19. P. Schweizer, "Physical instantiation and the propositional attitudes," *Cognitive Computation*, vol. 4, pp. 226–235, 2012.
20. D. Kazakov, "The self-cognisant robot," *Cognitive Computation*, vol. 4, pp. 347–353, 2012.

21. P. Wang, “Embodiment: does a laptop have a body?,” in *Proc. of AGI Conference*, (Arlington, Virginia, USA), pp. 174–179, March 2009.
22. F. de Saussure, *Grundfragen der allgemeinen Sprachwissenschaft.* Berlin: de Gruyter, 1915. ed. Charles Bally unter Mitw. von Albert Riedlinger, translator Herman Lommel (2001).
23. L. Wall, *Perl Language Reference Manual - for Perl version 5.12.1.* Network Theory Ltd., 2010.
24. R. Burch, “Charles Sanders Peirce,” in *The Stanford Encyclopedia of Philosophy* (E. N. Zalta, ed.), Aug. 2010. online; accessed Apr-2015.
25. E. Nozawa, “Peircean semeiotic - a 21st century scientific methodology,” in *Proc. of the Int. Symp. on Collaborative Technologies and Systems*, (Orlando, FL, USA), pp. 224–235, May 2007.
26. R. Barthes, *Elements of Semiology.* London: Jonathan Cape, 1964. trans. Lavers A and Smith C (1967).
27. B. B. Rieger, “Computing fuzzy semantic granules from natural language texts. A computational semiotics approach to understanding word meanings,” in *Proc. of the IASTED Int. Conf. on Artificial Intelligence and Soft Computing* (M. Hamza, ed.), (Honolulu, Hawaii, USA), pp. 475–479, August 1999.
28. P. Berkhin, “Survey of clustering data mining techniques,” tech. rep., Accrue Software, Inc., 2002.
29. P. Ernest, “A semiotic perspective of mathematical activity: The case of number,” *Educational Studies in Mathematics*, vol. 61, pp. 67–101, Feb. 2006.
30. J. Goguen, “An introduction to algebraic semiotics, with applications to user interface design,” in *Computation for Metaphor, Analogy and Agents* (C. Nehaniv, ed.), pp. 242–291, Springer Lecture Notes in Artificial Intelligence, 1999.
31. K. Lengnink and D. Schlimm, “Learning and understanding numeral systems: Semantic aspects of number representations from an educational perspective,” in *Philosophy of Mathematics: Sociological Aspects and Mathematical Practice* (B. Löwe and T. Müller, eds.), pp. 235–264, London College Publications, 2010.
32. P. Thagard, “Abductive inference: from philosophical analysis to neural mechanisms,” in *Inductive Reasoning: Cognitive, Mathematical, and Neuroscientific Approaches* (A. Feeney and E. Heit, eds.), (Cambridge), pp. 226–247, Cambridge University Press, 2007.
33. B. Russell, *The Problems of Philosophy.* Oxford University Press, 1959.
34. T. M. Mitchell, “The need for biases in learning generalizations,” in *Readings in Machine Learning* (J. W. Shavlik and T. G. Dietterich, eds.), pp. 184–191, Morgan Kauffman, 1980.
35. S. J. Nasuto, J. M. Bishop, E. B. Roesch, and M. C. Spencer, “Zombie mouse in a Chinese room,” *Philosophy & Technology*, vol. 28, pp. 209–223, June 2015.
36. P. Konderak, “On a cognitive model of semiosis,” *Studies in Logic, Grammar and Rhetoric*, vol. 40, pp. 129–144, 2015.
37. S. Colton, “Refactorable numbers: a machine invention,” *Journal of Integer Sequences*, vol. 2, 1999.
38. A. D’Ulizia, F. Ferri, and P. Grifoni, “A survey of grammatical inference methods for natural language learning,” *Artificial Intelligence Review*, vol. 36, no. 1, pp. 1–27, 2011.
39. Z. S. Harris, “Distributional structure,” *Word*, vol. 10, pp. 146–162, 1954.
40. S. M. Lamb, “On the mechanization of syntactic analysis,” in *1961 Conference on Machine Translation of Languages and Applied Language Analysis, Vol. 2 of National Physical Laboratory Symposium No. 13*, (London), pp. 674–685, Her Majesty’s Stationery Office, 1961.
41. Z. Solan, D. Horn, E. Ruppin, and S. Edelman, “Unsupervised learning of natural languages,” *Procs. of the National Academy of Sciences*, vol. 102, no. 33, pp. 11629–11634, 2005.
42. T. Ionin and O. Matushansky, “The composition of complex cardinals,” *Journal of Semantics*, vol. 23, pp. 315–360, Nov 2006.
43. R. Sinatra, D. Condorelli, and V. Latora, “Networks of motifs from sequences of symbols,” *Phys. Rev. Lett.*, vol. 105, p. 178702, Oct 2010.