Formal theory of thinking

Anton Venglovskiy [0000-0001-5780-6572]

anton.venglovskiy@gmail.com

Kyiv, Ukraine, 2021

Abstract. The definition of thinking in general form is given. The constructive logic of thinking is formulated. An algorithm capable of arbitrarily complex thinking is built.

Keywords: artificial general intelligence, thinking algorithm, theory of thinking.

1 Introduction

In this manuscript, I would like to share with you my invention, its a formal, axiomatic theory of thinking, within which I describe a thinking algorithm. Since I firmly believe that my modest invention is theoretically and practically useful, I decided to write this manuscript. Below you will find an introductory commentary, my theory and algorithm.

First of all, it should be noted that from the point of view of my theory, intelligence, and thinking are conceptually independent of each other, this means that intellect and thinking can exist one without the other, they are different phenomena, from different levels of abstraction. I algorithmize precisely the intuition of thinking, but not the intelligence.

As examples of what I mean by thinking, one can point to such hi level phenomena as the evolution of a personal or collective worldview, as well as the evolution of society, the evolution of language, science, art, and the evolution of human knowledge in general and in particular. In a more down-to-earth sense, the existence of thinking as an intuitively obvious phenomenon can be seen in the process of solving some difficult problem with previously unknown principles. By experience, it will be a spontaneous, evolutionary process in which the complexity of reasoning at the beginning of the solution and at its end will be different, at that intuition tells us that the achievement of a solution in problems of this sort will be accompanied by an increase in the complexity of actions and reasoning, since the invention of new principles of action and reasoning will be required. The listed processes demonstrate well the universal landscape of human thinking—productivity, logicality, spontaneity, and increasing complexity. Thus the main thesis of my theory is that thinking and the evolution of complexity are one and the same phenomenon. It is important to emphasize exactly the evolutionary nature of the thinking processes, which means the thinking process retains logical coherence at various levels of abstraction over time. The algorithm that I propose is an ideal model

of the complexity evolution process. The process is formally structured as a hierarchy of some abstract interconnections, but its structure is devoid of other meanings than complexity, relative significance, and diversity, roughly speaking - it is a meaningless evolving complexity in its pure and explicit form. Using the proposed algorithm, it is possible to model a large variety of evolutionary processes of any given complexity.

You can pay attention to the fact that my definition of thinking includes processes in which there is no human intelligence, for example, the evolution of an embryo is clearly different from the evolution of scientific knowledge, although both are thinking according to the definition. It's obvious that these two types of evolution are structured differently: the evolution of the embryo is structured by biochemical rules, whereas the evolution of scientific knowledge is structured by the rules of intelligence. The advantage of my approach is that "pure thinking" implemented in the algorithm can be structured in any desired way, including it can be structured as intelligence. The "pure" thinking process that I propose can be interacted with at different levels of abstraction, and the nature of the interaction will determine the structure of thinking. If the interaction (with the help of a feedback loop) is intellectual in nature, then in the process of evolution the computational process of thinking will be structured as intelligence, with all the ensuing consequences.

2 Experiment

I propose to verify the validity of my assumptions of thinking by solving some fairly complex inventive problem. For example, there is the so-called "the hardest logic puzzle ever" by George Boolos, and once I solved this puzzle and I can describe my experience. In the proposed task, you need to come up with the no more than three questions to three sober-minded Gods - one always tells the truth, the other always lies, and the third answers truth or lie randomly - and ask these questions to one or more from Gods in order to find out who is who; with the additional condition that the Gods gives their answers Yes / No in an unknown language. Tackling the solution for the first time, I exhausted possible options for questions in a few days and did not come close to a solution, and in general, it seemed that there could be no other questions in principle, a strange feeling: "How it happened if there is a solution, why can't I think of?". Having failed himself I tried to find a theory or algorithm for solving similar problems, but did not find it. So, unable to cope with the task, I postponed the task. After a year's break, I returned to the task again and with a fresh mind noticed a way to complicate the questions for Gods, and the solution quickly opened! Comprehending this case can give a general theoretical recommendation - you need to make the questions fundamentally more complex in all senses. The questions that I could come up with in the first iteration of the solution were too simple, and the main difficulty was just how to make more complex questions virtually from nothing. So, to make a more complex questions, I performed two universal steps: first, I added new, random properties at all levels of abstraction to the existing questions, and in the second step I generalized the results of the first step, and did this in a circle many times until the generalized conception of questions reached complexity enough for solution.

Comparing the questions I could ask in the first iteration of the solution with the questions I invented in the second iteration, there can be no doubt that the complexity of the questions has increased. And in all problems for which there is no typical solution I always observe the same general scheme - a transition from less complexity of logic to greater complexity is required, and this transition takes place in thinking, and this is achieved through random modifications (e.g. by expanding the context) and subsequent generalization. Therefore, it seems plausible to suggest that the defining feature of thinking is the ability to create logically organized content of the necessary complexity. And whether this is so, you can check on the experience of solving the puzzle.

3 Logic of thinking

About **thinking in general**. When considering thinking, it is impossible to abandon four fundamental characteristics: productivity, logicality, spontaneity, and complexity. If even one of these characteristics is absent, then there is no thinking. Therefore, thinking in general form is the spontaneous production of logically organized content of a necessary complexity. To model thinking, it is necessary to constructively formalize the enumerated fundamental characteristics.

Logic in general. Logic implies the main, the secondary and their dynamics. There is no logic without connections and levels of abstraction. Consequently, any logically organized content can be structured as a hierarchy of interconnections, where content from higher hierarchies is more significant than content from lower ones. Transformations that preserve the content hierarchical ordering property are logical operations. Any defined set of possible sequences of logical operations is some logic. Logic will be spontaneous if the set of possible sequences of logical operations is random.

The definition of logic given above allows you to explicitly define a special set of structured objects so that any mappings on this set are logical operations. The hierarchy of interconnections can be written literally, for example, in the following form: ((ab)c) is a rooted tree, the levels of which correspond to the levels of significance of the content, and each node of the tree contains a collection of arbitrary identifiers - which is interconnected content. For ease of writing, I skip separating characters when writing identifiers, so *each letter is a separate identifier*. The collection can be empty and the identifiers can be repeated. For example, there may be trees like this: (a), ((aa)(aa)), ((a)(b)), (ab(cd(efg(a)))), (ab(ab)(ab)(aabb)), (a(b(cccc(dd)(ee)))) and the like. It is convenient to operate with such trees as strings, so from now on I will speak simply - strings. Isomorphism's of strings are identical, for example, ((a(b))c) ≡ (c((b)a)). On the set S of all possible strings of the specified type, any operations of the type $S \rightarrow S$ will be logical operations, so any sequence of string transformations will correspond to some kind of logic. This logical universality of content from S is suitable for the logic of thinking in general.

The logic of thinking. The logic of thinking implies complexity and its dynamics. There is no thinking if there is no possibility of transition from less complexity to more complexity and vice versa. Thus, in order to set the logic of thinking on the set S, it is necessary to determine the complexity on the set S. Complexity can be defined explicitly: firstly, there must exist prime objects and complex ones, secondly, any complex object can be decomposed into prime objects, thirdly, a complex object has properties that its decomposition does not have, and fourthly for prime objects, the decomposition is empty. This corresponds to the natural intuition about complexity: if a thing can be taken apart into simple parts and, if the thing and a set of its spare parts are not the same, then the thing is undoubtedly complex. The definition of complexity thus given requires a decomposition procedure, and such a procedure can be given on the set S, A: $S \to 2^S$, if $s \in S$, $A[s] = \emptyset$ then s is prime string; else, if $A[s] \neq \emptyset$, $\forall a \in A[s]$, $A[a] = \emptyset$ then s is complex string, besides $\forall s_1, s_2 \in S$, $s_1 \neq \emptyset$, $s_2 \neq \emptyset$, $s_1 \neq s_2 \Rightarrow A[s_1] \neq A[s_2]$; further I call the decomposition procedure A an Abstraction operator since it is based on the operation that explicitly deduces the main and discards the secondary, which is abstraction by definition. Well if string decomposition is set, then complexity is set, but this is not enough for the logic of thinking, since the definition requires the possibility of increasing complexity, and this requires another procedure on S, D: $S \to S$ that can make complex strings from simpler or primary ones, $\forall s \in S$, $C[s] < C[D[s]]$, C: $S \to \mathbb{N}$, $C[s] \equiv |A[s]|$, C calculates the complexity value, besides $\forall s_1, s_2 \in S$, $s_1 \neq \emptyset$, $s_2 \neq \emptyset$, $s_1 \neq s_2 \Rightarrow D[s_1] \neq D[s_2]$. Operator D can also be given on the S; further I call D the Deduction operator because it is based on the natural deductive rule. Thus, two corresponding operations: Abstraction and Deduction gives the logic of thinking on the S.

Operation of Abstraction. Acts on the principle of detecting a common property for a group of similar objects. On the set S, this principle can be expressed literally: $((ab)(ac)) \Rightarrow (a(bc))$. The general letter "a" can be put out from the parentheses, thus raising its level in the abstractions hierarchy, the letter "a" has become a generic feature for everything lower in the hierarchy, and the remnants are merged in common parentheses because this is a set of specific features that were discarded when abstracting a generic feature "a". After the action of the operation, the construction of the string is simplified, and the loss of insignificant information (compression) occurs, which is also characteristic of natural generalization procedures. The rule can be generalized. Firstly, let's define the concept of a substring, since a string is a rooted tree, then a substring is any subtree. For example, the string (ab(cd)(ee(bb))) contains four following substrings: (ab(cd)(ee(bb))), (ee(bb)), (cd), (bb). Write $s_1, s_2 \in S$, $s_1 \subset s_2$ means that s_1 is a substring of s_2; $\forall s \in S$ notation $s \subset s$ is valid. Second, let's define the concept of string elements: for any string or substring, the collection of its elements is the union of root node identifiers and the root node substrings, so the collection is heterogeneous and a single element can be an identifier or a string. The collection of elements for the string s will be denoted by E_s, the collection is multiset. For example, consider the string s = (aabb(cd)(ee(bb))((a)(b))) its elements will be E_s = {a, a, b, b, (cd), (ee(bb)), ((a)(b))}. Write $s, s_1 \in S$, $s_1 \in E_s$ means that s_1 is an element of s.

To denote a subcollection, the following notation will be used: $E_1 \subset E_s$, and the notations $s_1 \in E_1$, $E_s \subset E_s$ will be valid. For collections, let's define two operations: ins[] and del[], inserting new elements into the collection and removing elements from the collection, respectively: $\forall s_1, s_2 \in S$, $ins[E_{s1}, E_{s2}] \Rightarrow E_{s1} = E_{s1} \uplus E_{s2}$, $del[E_{s1}, E_{s2}] \Rightarrow E_{s1} = E_{s1} \setminus E_{s2}$. The collection can be specified explicitly $E_1 = \{s_1, s_2..s_n\}$. Now let's write the Abstraction rule in general form: $\forall s_o \in S$, $\forall s_b \subset s_o$, $\forall s_1..s_n \subset s_b$ such that $\forall i \in [1..n]$, $s_{[i]} \in E_{sb}$, $\forall E_1 \subset \bigcap_{\forall i \in 1..n} E_{s[i]} \Rightarrow ins[E_{sb}, E_1]$; $del_{\forall i \in 1..n}[E_{s[i]}, E_1]$; $\exists s_a$ such that $E_{sa} = \uplus_{\forall i \in 1..n} E_{s[i]}$, $ins[E_{sb}, \{s_a\}]$; $del[E_{sb}, \{s_1..s_n\}]$. Semicolons separate sequential operations, a record of the form $\forall i \in [1..n]$ means a cycle by i. The decomposition procedure A[s] sequentially and recursively goes through all possible applications of the abstraction rule for a given string and thus finds a decomposition into prime strings.

Let's look at an illustrative example. The abstraction rule is valid for any substrings, any given string. If any group of substrings is at the same level of the hierarchy, for example, (...()()...), (...()()()...), (...()()()()...), ..., (...(...()()...)...), (...(...()()()...)...), ..., and if any combination of substrings contains the same content, for example, like this (...(ab)(aabb)...), or this (...(ab)(aabc)(aabbcc)...), then any piece of matching content can be put out from the parentheses, for example like this ((ab)(aabc)(aabbcc)) ⇒ (ab(ac)(aabbcc)), or like this ((ab)(aabc)(aabbcc)) ⇒ (a(babcabbcc)), or this ((ab)(aabc)(aabbcc)) ⇒ ((ab)abc(aabc)), ((ab)(aabc)(aabbcc)) ⇒ (ab(aabccc)) and there are still many options left. The content that is putted out from the parentheses is underlined. As you can see, after putting out the general content, the remnants in the parentheses must be merged. You can put out from the parentheses not only identifiers, but also substrings, for example, (((a(b))(c(b))) ⇒ ((b)(ac)), and if there are empty parentheses, they must be discarded, for example, (((a(b))(a(b))) ⇒ (a(b)()) ≡ (a(b)). Since a string can be simplified in many alternative and mutually exclusive ways, the Abstraction rule must be performed sequentially, that is, in one step, you can make only one simplification for any one group of parentheses and one selected piece of content. Using the Abstraction rule, for any given string, you can sequentially construct the set of all possible simplifications in all possible alternative ways, using recursion. For example, consider the string ((a)(bc(a))(bc(a))), and write down all possible simplifications: on the first iteration its be ((a)(bc(a))(bc(a))) ⇒ (b(a)(cc(a)(a))), ((a)(bc(a))(bc(a))) ⇒ (c(a)(bb(a)(a))), ((a)(bc(a))(bc(a))) ⇒ (bc(a)((a)(a))), ((a)(bc(a))(bc(a))) ⇒ ((a)(a)(bbcc)), ((a)(bc(a))(bc(a))) ⇒ ((a)bc(a)), now apply the Abstraction rule to the results obtained at the first iteration, where possible (b(a)(cc(a)(a))) ⇒ (b(a)(cca)), (c(a)(bb(a)(a))) ⇒ (c(a)(bba)), (bc(a)((a)(a))) ⇒ (bc(a)(a)), ((a)(a)(bbcc)) ⇒ (a(bbcc)), ((a)bc(a)) ⇒ (abc), again apply the Abstraction to the results of the second iteration (b(a)(cca)) ⇒ (ab(cc)), (c(a)(bba)) ⇒ (ac(bb)), (bc(a)(a)) ⇒ (abc), that's all, there are no more options, the procedure recursively converged. Thus, with the help of recursion, we have decomposed the source, complex string into many simple and prime strings. Recursive decomposition of the source string into a set of prime ones using the Abstraction rule, I will call the Abstraction operator A[s], to summarize, we write, A[((a)(bc(a))(bc(a)))] = {(b(a)(cc(a)(a))), (c(a)(bb(a)(a))), (bc(a)((a)(a))), ((a)(a)(bbcc)), ((a)bc(a)), (b(a)(cca)), (c(a)(bba)), (bc(a)(a)), a(bbcc)), (abc), (ab(cc)), (ac(bb)), (abc)},

the specified multiset of Abstraction results contains all possible generalizations of the source string.

Deduction operation. Adds new constructive properties to the source hierarchy, extends, and always complicates the interconnections hierarchy. A strict description of the Deduction algorithm is unreasonably cumbersome, so I will limit it to an intuitive description with examples. On set $\mathbb{S}$, there is a simple and natural way to define Deduction. First, note that strings can be written in a more compact form if repeating elements are written using a multiplier prefix, for example, (aa) ≡ (2a), ((a)(a)) ≡ (2(a)), ((aa)(aa)) ≡ (2(2a)), (aa(bb(ccc)(ccc))(bb(ccc)(ccc))) ≡ (2a2(2b2(3c))), in this notation any string is unambiguously unpacked from left to right. The Deduction rule duplicates all elements of the source string and its substring, any given number of times, and is general case defined as follows: (a) ⇒ {(2(2a)), (3(3a)), (4(4a)), ...}, that is, from any source string, you can get an infinite number of Deductions that will differ in the number of duplicates of the source elements. For practical purposes, in order to write down the algorithm, the number of duplicates must be fixed, so I define the Deduction operator as $D: \mathbb{S} \rightarrow \mathbb{S}$, D[(a)] = (2(2a)). More examples: D[(aa)] = (2(4a)), D[((a)(b))] = (2(2(2a)2(2b))), D[(a(b(c)))] = (2(2a2(2b2(2c)))) and the like. As you can see, even if the source string is prime relative to the Abstraction rule, then its Deduction will always be complex, that is, it will have a non-empty decomposition into prime strings, for example, A[D[(a)]] = A[(2(2a))] ≡ A[((aa)(aa))] = {(aa), (a(aa))}, from this example it is obvious that the source string (a) after Deduction acquired fundamentally new constructive properties that Abstraction detects. In practice, the variety in decomposition into prime strings depends on the number of duplications.

Complexity calculation. Let's return to the definition of complexity. As mentioned above, complexity requires a decomposition operator, which parses a complex string into many simple ones, with some refinements (see above). We constructed such an operator and called it Abstraction. Next, you need to pay attention to the fact that operator A is a logical operation, according to the definition of logic given above. Therefore, we are talking specifically about the logical complexity of strings, and not about some other complexity that might not be logical. In addition, A is a generalization procedure, and as a result of decomposition, we get many generalizations of the original string and not just some simple parts that might not be generalizations. To summarize - A sets logical complexity and is a generalization procedure in terms of its mode of action. I also believe that A is the only possible operator specifying logical complexity on $\mathbb{S}$, but I have no proof.

There is a well-known heuristic metric for complexity: the number of components is proportional to complexity. Everyone has heard about the myriad of parts in a spaceship, and everyone can imagine how relatively few parts there are in a car, and even fewer in a bicycle. This intuition can be extended to more abstract things, for example, to blueprints and descriptions - complex blueprints and descriptions contain many different design concepts necessary for their draw-up. And it is obvious that complex descriptions will contain more concepts than simple descriptions. Further, the complexity metric can take into account either all components - including repeated ones,

or only unique components. In our case, only unique components are taken into account. So, relying on the heuristic indicated, the complexity of a string is equal to the number of unique prime strings in its decomposition. In turn, the complexity of prime strings is the same. The last point is not intuitive, since prime strings can have different structures and different lengths, for example, there can be short strings with a shallow hierarchy and there can be long strings with a deep hierarchy. It seems that prime strings must also have some kind of relative complexity - intuition makes us suspect the existence of some kind of internal logic in prime strings. However, from the point of view of the decomposition procedure, the complexity for simple strings is not defined, and within the framework of the formalism there can be no additional assumptions about what the complexity of prime strings could mean; these are elementary, although different objects. Therefore, as a matter of common sense, we will assume that the complexity of prime strings is the same. Based on this understanding, the complexity metric can be further improved. With the help of the Deduction operator, we could make a complex string out of a prime one - it would be some new object, and its complexity would be proportional to the source string complexity. In this way, it is possible to formulate a more advanced complexity metric that will be consistent with the intuition of "internal logic": instead of the value |A[s]| you can calculate the complexity of Deduction |A[D[s]]|, this is a kind of potential complexity, and it is defined for any string from S. Moreover, the potential complexity can be calculated with arbitrary precision, for this you need to alternate the steps of Deduction and Abstraction as required times recursively.

Before summing up this paragraph, it is necessary to clarify that earlier - in the illustrative example of how the Abstraction operator works, I included in the result all the strings obtained during the decomposition, this was done for ease of understanding, but in reality the result A should contain only unique prime strings. So let's summing, complex things have more possible generalizations than simple things. The complexity of things implies generalization. The logic of an object is more complex the more different generalizations can be established for a given object. Thus, the Abstraction operator naturally defines the complexity on the set S. The more prime strings in the decomposition of source string using the Abstraction operator, the more complex the source string is. Let's define complexity as C[s] ≡ |A[s]|, where |A[s]| is the number of unique prime strings in the set of Abstraction results. The operator C can be used to express the main property of Deduction, C[s] < C[D[s]], which is obviously by construction. The complexity of all prime strings is the same and does not depend on their size and hierarchy, for example, C[(a)] = C[(a(b)(c))] = C[(a(b(c))(c(a)))]. However, for prime strings, you can define their potential complexity through Deduction, C[D[(a)]] < C[D[(a(b)(c))]] < C[D[(a(b(c))(c(a)))]], in which case the potential complexity will depend on the size and structure of the source prime strings.

Thinking process. On the set S, Abstraction and Deduction procedures are given, as well as complexity is defined. Thus, the logic of thinking is set. Now within the framework of the described formalism, it is possible to manipulate logical and complex objects that are represented by strings, and most importantly, it is possible to make logical transitions from less complexity to greater complexity, which is a defining feature of thinking, according to our definition. Since the specific level of human

thinking complexity is unknown, and since it seems that the human thinking complexity is very great but still finite, then only an infinitely complex computational process can simulate human thinking for sure. Thus computational process satisfies the criteria for real thinking if it can create content of arbitrary complexity. To formally write such a process, an additional string concatenation operator is required. For syntax uniformity, the concatenation operator will be written as parentheses ({...}), the concatenation operator can be applied to any set of strings and result in one string, for example, (A[D[(a)]]) = (A[((aa)(aa))]) = ((aa)(a(aa))). As you can see, many of the two Abstraction results have been concatenated into one string. So, the thinking process in general form: $t_n = (A[D[t_{n-1}]])$; $t_0 \in S$. This recursive function produces logical, unique and arbitrarily complex content in any quantity from any nonempty starting value.

Spontaneity. Thinking is spontaneous, no one can predict their next thought. In thinking, temporal trends can be identified, so you can predict the direction of thoughts to some extent, for example, you can roughly guess what you will think tomorrow, but such predictions are mostly wrong since the origin of trends is also accidental. Collective forms of thinking, such as the development of science or the history of society, are also clearly random. No one can reliably predict history or how this or that scientific theory will develop. Spontaneity is the guarantee of any fundamental novelty and this property is not reducible to the effect of a set of external factors, but it is inherent in thinking, which is obviously observed in closed subjective thinking. Thereby the computational process of thinking must be algorithmically random. In a strict sense, I believe that the set of all possible substrings that the function t_n enumerates at the abstraction stage is undecidable, and therefore the function t_n is algorithmically random, but I have no proof.

Finale stage of reasoning, given spontaneity, the function t_n fully satisfies the definition of thinking, all properties of thinking are constructively formalized. This function produces spontaneous, logically organized content of any given complexity. Moreover, the complexity is defined explicitly and constructively, and it can be calculated and evaluated. The logical organization of content is guaranteed by the syntax and the Abstraction operator, which iteratively generalizes the logic of the produced content. Thus, the function t_n represents an algorithmic form of thinking in its pure, ideal form. Calculating this function is thinking.

Adjustment of generalizing ability and complexity of thinking. As you can see from the construction, the complexity of the content produced by the t_n function increases exponentially, this is impractical. However, there is a natural way to regulate the generalization and complexity of content over a wide range. Any fixed level of complexity and abstractness of content can be maintained. Due to the specific action of Abstraction, highly organized content floats to the upper levels of the hierarchy, while at the lower levels of the hierarchy, more chaotic content remains, that is, the content is ordered by the level of significance, generality and logical organization. Therefore, it is possible to discard the lower, insignificant levels of the hierarchy, that is, you can perform additional generalization of content using a cruder method. For example, from the string (abc(de(fk(gh)(cn(rt))))) you can extract the most significant part by cutting off deep nesting levels, like this (abc(de(fk(gh)(cn(rt))))) ⇒ (abc(de(fk))), in

showed case a relatively insignificant fragment (gh)(cn(rt)) was removed. The procedure for removing insignificant content can be built into the t_n function after the Abstraction stage, like this $t_n = (\text{Truncate}[A[D[t_{n-1}]], d])$, where d is the depth of the removed content. In addition to deletion of deep nesting, at the Abstraction stage it is also possible to selectively leave the Abstraction results, for example, it may make sense to leave only prime strings since they are the most generalized, and filter out the rest. In general, it is possible to filter the results of Abstraction according to their statistical significance for each specific practical problem.

Practical use. In order for the described formal thinking process to acquire a meaningful character, it is necessary to feed it a meaningful external signal, encoded with strings, according to the following scheme: $t_n = (A[D[((t_{n-1})(\text{InputSignal}_n))]])$, if a certain meaning is encoded in the input signal, then the behavior of the t_n process will also have a certain meaning. The t_n process thinks abstractly in its own internal "language", in order to understand the content of such thinking, it is necessary to build a model of a "common language" with the help of a feedback loop.

Intelligence. In the context of thinking, I consider intelligence as an epiphenomenon. That is, intelligence is a dependent and optional form of the general process of thinking. This form of thinking can arise in response to a specifically intellectual interaction with the general process of thinking. In order for the process of thinking t_n to acquire an intelligence character, it must be structured in terms of the rational achievement of goals through interaction with an external intelligent system. The combination of t_n with an external intelligent system in a feedback loop will be a kind of Artificial General Intelligence.

Consciousness. The idea that an algorithm can somehow have consciousness is wrong. The algorithm consists of parts and exists in steps, while consciousness, on the contrary, does not consist of parts and does not exist in steps. "Algorithmic" and "unconscious" are synonymous. All so-called functional manifestations of consciousness belong exclusively to the sphere of thinking. The correct understanding is that there are various forms of thinking, in particular, there is conscious thinking - like in humans, and algorithmic thinking, that is, the unconscious. And the only question is whether an algorithmic form of thinking is possible, and this manuscript answers this question positively.

This position allows us to make several constructive assumptions about conscious thinking. Although conscious thinking cannot be a discrete entity, however, it is a process. This means that if an analog, stochastic, complex process is physically possible, and isomorphic to the discrete process t_n, then it is likely to be conscious in the human sense. To confirm the last assumption, a natural experiment is possible, one could look for patterns of complexity oscillations (corresponding to the Abstraction-Deduction cycle) on the active neurological substrate and evaluate the correlation with the presence of consciousness. If there is a significant correlation, then the nervous system is likely to simulate an analog version of t_n of some complexity, which would indirectly confirm that similar processes on other physical substrates will also be conscious.

In addition, we can offer a plausible version of what the analog implementation of t_n should be. But first I want to clarify that we are talking specifically about conscious thinking, that is, about the process of the evolution of complexity with some additional specificity - such that makes this process a good candidate for the assumption of the experience of sensation. The behavior of the algorithm is deterministic - the future is predetermined, although unknown in advance (weak spontaneity), which means that in algorithmic thinking there is no independent subject, algorithmic evolution does not make a real choice. It would be a completely different matter if there were strong spontaneity, that is, there would be an indeterminism in the process of complexity evolution. In the case of indeterminism, the future is not only unknown but also not predetermined, which is a necessary prerequisite for the existence of an independent subject with free will. That is, thinking + indeterminism = independent, subjective thinking because there are all the necessary attributes of the subject: independence of behavior (strong spontaneity) and tendentiousness of behavior - one's own motives, as well as the ability to comprehend one's own behavior at different levels of abstraction. Such a process is apparently embodied in the human nervous system, and one can roughly estimate its structure. For example, on a neurological substrate, random, self-sustaining feedback loops of evolving complexity could emerge from spontaneous group interactions of cells, which would by definition be a form of thinking, with physically independent cell activity guaranteeing indeterminism in the behavior of such loops. Hypothetical self-sustaining feedback loops would be free to influence their own evolution to their own advantage, and thus influence the physical behavior of the cellular ensemble involved in the loop. It would be a kind of magic since we are talking about a feedback loop with a purely virtual subject, which does not depend on the physical substrate and at the same time can influence the physical substrate through the feedback in which it is involved. This configuration limits from below the range of processes for which the assumption of sensation experience is plausible.

4 Conclusion

I have no conclusions other than the thinking algorithm. The following is just random text for the sake of styling.

The main proposal is to free thinking from any kind of a priori phenomenology, from things, actions and meanings. Can is seen that thinking in general form does not need these concepts. Things and the logic of things are secondary products of thinking. For an accurate definition of thinking, the idea is important that the mind is a very high-level process, it does not depend on the meaning of what it creates, and therefore does not need the concept of meaning in its own definition.

From the point of view that thinking does not depend on meaning, one can proceed to the formal definition of thinking through complexity and then to its algorithmization - to a universally thinking algorithm. To do this, it should be noted that thinking has a universal product - this is complexity. All human theories of things are complex. This means that we can assume that the mind achieves its goals at the level of complexity of things, and the meaning of things is a by-product, and therefore thinking does not

depend on meaning. The main question to which thinking answers (without meaning) is how to make other fundamentally new, more complex things from a given, simple thing. Thus, all transformations of thought are transformations of the constructive complexity of things. However, for an outside observer, the phenomenology of thinking can look infinitely vast, because transformations at the level of complexity create a diverse, new logic of things.

In turn, the transformations of things at the level of complexity can be algorithmized in a general and explicit form. And the resulting algorithm, therefore, will be an accurate, constructive definition of mind in a general form. The main task of the algorithm is to create a new logic of things, of any required complexity. For this, the concept of logic is generalized and things are divided into logical and illogical. The algorithm creates only logical things and therefore they can be investigated and comprehended in concepts corresponding to the logical complexity of things.

The algorithm can be interacted with, and the algorithm is able to independently comprehend the nature of interaction at various levels of abstraction in the subjective system of concepts, which, along with the ability to evolve, is a prerequisite for high-level communication and intelligence. Intelligence is seen as a form of thinking that arises in response to meaningful interaction. In other words, any meaningful interaction, as a result of generalization, structures the subjective logic of the algorithm as the logic of intelligence. Thus, the initial assumption that thinking in its pure form does not need meaning does not exclude the possibility of meaning and intelligence and even turns out to be a prerequisite for their emergence and existence.