ISSN 1995-0802, Lobachevskii Journal of Mathematics, 2021, Vol. 42, No. 10, pp. 2358–2363. © Pleiades Publishing, Ltd., 2021.

Is Genome Written in Haskell?

S. V. Kozyrev*

(Submitted by G. G. Amosov)

Steklov Mathematical Institute of Russian Academy of Sciences, Moscow, 119991 Russia

Received April 5, 2021; revised April 13, 2021; accepted April 22, 2021

Abstract—This paper continues the discussion of genome as a functional program and biological evolution as learning for functional programs. Here we discuss gene regulation as monadic computation, in particular we consider Lac operon as an analogue of IO monad. This supports the idea of genome as a program written in Haskell-like programming language where recursive applications of lists of functions (genes) express parallel processes in a cell and gene regulation can be described by monadic computations.

DOI: 10.1134/S1995080221100127

Keywords and phrases: *learning theory, statistical mechanics, evolution theory, functional programming.*

1. INTRODUCTION

In this paper we continue the discussion started in [1–3] of the approach *"genome as a program"* using functional programming. We consider a genome as a functional program, say a lambda term or a program written in Haskell-like language. Functional programming is characterized by high degree of parallelism, simple system of states allows error control and easy modification of programs [4, 5]. In biology we observe high parallelism of processes in cells and in evolution, random modifications of genetic program in evolution usually do not break the program immediately. In the approach under consideration this parallelism is discussed as a manifestation of the functional style in which genomes as programs are written and genes are considered as functions in functional programming. The problem of biological Darwinian evolution in our approach is described by learning theory for functional programming.

Our aim is to investigate the syntax of the language in which genomes as programs are written. An important question in this approach is the description of gene regulation. Lac operon gives an example of gene regulation. Operon is a group of simultaneously transcribed genes with the same promoter and terminator of transcription. Lac operon contains CAP binding site, promoter, operator, three structural genes and termination of transcription (as situated in the DNA). Structural genes encode two enzymes and transport protein for lactose. Transcription is initiated depending on concentrations of lactose and glucose. For this aim two proteins binding to regulatory segments of the operon are used: lac repressor (sensor of lactose) and Catabolite activator protein (CAP), sensor of glucose. Presence of two sensors allows to start expression of structural genes in the case of simultaneous presence of lactose and absence of glucose.

We propose to describe gene regulation using the notion of monad of functional programming. Monadic values express the idea of "value in context" and application of a monadic function put values in context (some "monadic laws" should be satisfied). The context in genomics can be described by a set of regulatory molecules for operon, i.e. for the lac operon the context describes which regulatory molecules (CAP and lac repressor) are bound to binding sites of the operon. In general, this gives a clue to the problem of description of syntax of the programming language of life (in which genomes are written).

*E-mail: kozyrev@mi-ras.ru

Exposition of this paper is as follows. In Section 2 we describe genomes as functional programs, defined by recursive parallel application of genes. In Section 3 we discuss lac operon as analogue of the IO monad in Haskell and gene regulation as monadic computation. In Section 4 we discuss Darwinian evolution as machine learning problem for functional programming with regularization by estimate for Kolmogorov complexity.

2. GENOME AS A PROGRAM

Let us describe our approach to biology. Biological molecules are linear polymers (proteins and nucleic acids) described by strings of symbols, state of a biological system is described by a set of strings with multiplicity (multistring). Multistrings (sets of molecules in a cell) are subject to transformations corresponding to genes. We consider the following transformations. Chemical reactions used in biological processes are transformations (operations of editing) of multistrings local in substrings (gluing, cuttings, substitutions of substrings, duplications of substrings, other editing operations for multistrings). Physical transformations (transfer of molecules to cells) are described by changes of multiplicities of strings in a multistring. Formation of complexes of molecules (used in particular for gene regulation) will be discussed in the next section as monadic operations. Genome is considered as a list of genes, each gene defines a transformation of multistrings performed by the protein (or RNA) encoded by the gene—editing operations and operations of transfer of molecules (actually some of transfer operations are performed by corresponding transport molecules encoded by some genes). Genes are also strings (of nucleotides) and genomes are multistrings.

Let us denote S the set of multistrings. Genome is a list $G = [g_1, \dots, g_n]$ of genes. Gene g_k defines a function—a transformation of multistrings $S \to S$ (we use the same notation for a gene as a string of nucleotides and as a function). The map g_k is multivalued in S (application of function g_k to object $v \in S$ is represented by a lambda–term where reduction can be made in multiple ways). In particular the map g_k may cut a string at the position of substring uv

$$u'uvv' \mapsto u'u + vv',$$

string may contain several such substrings and g_k can act on different strings in a multistring.

Genes as transformations operate in parallel thus genome as a program is highly parallel. We describe genome as a functional program $\widetilde{G}$ by recursive application of the genome as a list of functions $G = [g_1, \dots, g_n]$, each of functions g_k is a transformation of multistrings $S \to S$

$$\widetilde{G} = \widetilde{G} \circ G = [\widetilde{G} \circ g_1, \dots, \widetilde{G} \circ g_n]. \tag{1}$$

Here the space of objects (where the functions are applied) is the space S of multistrings, g_k are genes, G is a genome as a list of genes, and $\widetilde{G}$ is a genome as a program. List $G = [g_1, \dots, g_n]$ of genes is a multivalued function $S \to S$: any function g_k in the list (which itself is multivalued) can be applied to an object $v \in S$.

Metabolic network as a reduction graph. Graphs are often used to describe functioning of genomes, in particular metabolic networks describe pathways of chemical reactions in cells, also gene co-expression networks are considered. For discussion of retinal evolution evolutionary networks are investigated (where cycles in networks describe effects of hybridization and horizontal gene transfer) [6, 7]. From the point of view of functional programming these networks can be naturally discussed as reduction graphs for lambda calculus. Let us put in correspondence to program (1) a graph constructed in the following way.

Let $v_0 \in S$ be a multistring (this multistring should be "reasonable" from biological point of view, it should contain molecules needed for operation of the genome and the cell and do not contain unreasonable molecules). Let us define a graph $\Gamma_{\widetilde{G}}$ for the program $\widetilde{G}$ as follows:

Step 0) We start construction of the graph from vertex v_0.

Step 1) Let us apply G to v_0, any $g_k \in G$ can be applied to v_0 (g_k itself is multivalued). Let us include to the graph all vertices obtained from v_0 by multivalued map G (we identify vertices which coincide as multistrings), the obtained vertices are connected to v_0 by edges.

Step 2 etc.) By recursion let us apply multivalued map G to obtained at the previous step vertices. Let us include to the graph $\Gamma_{\widetilde{G}}$ all vertices and edges obtained in this way (again, we identify vertices

which coincide as multistrings and connect the obtained vertices to predecessors by edges). Iteration of the process gives graph $\Gamma_{\widetilde{G}}$.

Some genes g_k in G correspond to "physical" transfer operations which change multiplicities of some strings in a multistring. These operations allow to close metabolic cycles in the graph $\Gamma_{\widetilde{G}}$.

To a gene g in the genome G we put in correspondence the pair of non-negative numbers $r_+(g)$, $r_-(g)$—transition rates of corresponding direct and reverse reactions. These rates define a system of kinetic equations for distribution functions on vertices of the graph $\Gamma_{\widetilde{G}}$—equations describe transitions with rates $r_+(g)$, $r_-(g)$ along and against edges corresponding to genes. Let us assume that for this system of kinetic equations there exists a unique stationary state $f_{\widetilde{G}}$, moreover the solution of the system tends to $f_{\widetilde{G}}$. The state $f_{\widetilde{G}}$ describes metabolism in the cell, this state is non-equilibrium, in particular currents in this state are metabolic currents in the cell.

Let us consider some linear functional $A(f)$ of distribution $f(v)$ on vertices of the graph. In particular: the functional of current along the edge v_1v_2 with rates r_+ and r_- along and against the edge (from v_1 to v_2 and against) which equals to $r_+f(v_1) - r_-f(v_2)$. Different edges v_1v_2 corresponding to the same gene may be related to the same reaction (in particular with different values of reagents). In this case to obtain the complete current one has to sum up values $r_+f(v_1) - r_-f(v_2)$ over all such edges. We will discuss functionals (in particular currents) $A(f_{\widetilde{G}})$ in the mentioned above stationary state.

Remarks. The program $\widetilde{G}$ is highly parallel. Correctness of operation of metabolic networks is related to Church–Rosser property for lambda-calculus (in different order of application of genes one can obtain the desired result).

The program $\widetilde{G}$ for a genome loops—this describes cycles in the metabolic network $\Gamma_{\widetilde{G}}$.

The stationary state $f_{\widetilde{G}}$ corresponds to a state (in the sense of functional programming) for a genome as a program $\widetilde{G}$. In particular it changes in the process of gene regulation.

Composition of multivalued maps G in recursive definition (1) coincides with composition of lists of functions in Haskell according to the definition of list as applicative functor. Action of multivalued map G on multistrings is more complicated (in particular genes g_k in G can be multivalued).

3. GENE REGULATION: LAC OPERON

Gene regulation in our picture acts as follows: changing values $r_+(g)$, $r_-(g)$ we will change the stationary state $f_{\widetilde{G}}$ and contributions to the functional $A(f_{\widetilde{G}})$ from different metabolic pathways. Physically (or chemically) gene regulation works by regulatory molecules (in particular for lac operon) which regulate expression of genes. Analogy in functional programming can be described by computations in a context and computations with effects. In functional programming this situation is described by introduction of monads, in genomics the context of computations is given by complexes of genomes with regulatory molecules.

Lac operon is an analogue of the IO monad in Haskell (input–output). In the genome as a program approach lac operon can be considered as one of genes g_n in (1). We propose for lac operon the following Haskell-like syntax

```
main = do
   glucose <- sensorofglucose
   lactose <- sensoroflactose
   if not glucose && lactose
      then return ( structuralgene1
                    structuralgene2
                    structuralgene3 )
      else return()
```

Here functions glucose and lactose return boolean values for operations of input–output (which check presence of glucose and lactose correspondingly) and structural genes 1, 2, 3 perform operations of expression of corresponding genes. Therefore this function in absence of glucose and presence of lactose expresses structural genes and for the case else performs return( ) i.e. returns empty tuple. Empty tuple acts as identity transformation in the space S of objects. Expression of structural genes can be understood as action of transformations performed by structural genes (by proteins encoded by these genes). Activation of the lac operon changes the graph $\Gamma_{\widetilde{G}}$ of the program and stationary distribution $f_{\widetilde{G}}$ (i.e. the distribution $f_{\widetilde{G}}$ and the graph $\Gamma_{\widetilde{G}}$ itself are context-dependent, the context at molecular level is given by binding of regulatory molecules to binding sites in corresponding operons in the DNA.

4. DARWINIAN EVOLUTION AS LEARNING

Following Alan Turing [8] we will consider Darwinian evolution by selection as machine learning, or generation of programs by data. This leads to formulation of learning problems for functional programming. We will use temperature learning (i.e. instead of risk minimization problem we will consider the corresponding statistical sum for Gibbs distribution where the Hamiltonian equals the risk functional of the learning problem plus regularization).

Temperature learning. Problem of machine learning is the problem of minimization over the space of parameters s of the sum of the loss (or risk) functional and the regularization functional

$$H(s) = R(s) + Reg(s) \to \min .$$

We omit in the above formula dependence of the functional on the training sample. Regularization is important to control overfitting. Overfitting means that the solution of the learning problem shows considerable dependence on the choice of a training sample, in this case one could obtain low value of the risk functional at a training sample and high value of risk at a control sample. The main approach to control overfitting is regularization to effectively reduce entropy of the space of parameters s, see in particular VC-theory [9].

Temperature learning is defined as follows: instead of minimization we compute the statistical sum ($\beta > 0$ is the inverse temperature)

$$Z = \sum_s e^{-\beta H(s)}.$$

In the zero temperature limit $\beta \to \infty$ problem of computation of Z becomes the problem of minimization of H (temperature learning becomes standard learning).

It is natural to expect critical behavior in a temperature learning problem which (taking in account the discussion after formula (4) below) we formulate as follows. Let us consider the functional $H(\alpha, s) = R(s) + \alpha\, Reg(s)$, $\alpha > 0$ and the corresponding statistical sum $Z_\alpha = \sum_s e^{-\beta H(\alpha,s)}$. Then for sufficiently large α the statistical sum should converge (with a suitable regularization functional) and for small α it diverges. We propose to consider the divergence of this statistical sum as a criterion of overfitting using the physical intuition—in the high temperature (small α) regime the statistical sum Z_α "melts" and becomes divergent—values of parameter s which contribute to Z_α are not restricted to a space of limited entropy (as in the regime with overfitting in the learning problem).

Application of ideas of machine learning to biological evolution takes the form of regularization of the corresponding learning problem by estimate of Kolmogorov complexity to control overfitting (i.e. the risk functional in the learning problem for evolution describes selection pressure and regularization allows to avoid overfitting). Statistical mechanical models and Gibbs distributions were discussed in relation to scaling in genomics and linguistics (the Zipf's law). In genomics, Koonin [6] discussed genome as a *"gas of interacting genes"*, the corresponding Gibbs distribution should explain scaling in sizes of families of paralogous genes, scaling in metabolic networks and networks of interacting genes (which look like scale free graphs). Manin [10] investigated a model of statistical mechanics with Hamiltonian equal to Kolmogorov complexity (*"Complexity as Energy"* approach) and claimed that Gibbs distribution of this model should give the Zipf's scaling law for distribution of words in texts (moreover the Zipf's law is obtained at the temperature of phase transition). In [1–3] it was discussed that these two approaches can be unified if one will consider biological evolution as a model of temperature learning with regularization equal to estimate for Kolmogorov complexity. In particular

the universality of scaling in genomics can be explained by universal regularization by complexity in corresponding learning problems. Minimization of Kolmogorov complexity in application to neural networks was discussed in [11].

Temperature learning for functional programs. We will consider biological evolution as action of "evolution program" $\widetilde{E}$ defined recursively by the list of "evolution genes" $E = [e_1, \dots, e_m]$ (list of operations of editing of genomes) in a way analogous to (1)

$$\widetilde{E} = \widetilde{E} \circ E = [\widetilde{E} \circ e_1, \dots, \widetilde{E} \circ e_m]. \tag{2}$$

Evolution transforms genomes to genomes (as multistrings), transforms rates $r_+(g)$, $r_-(g)$ for genes in genomes, the stationary state $f_{\widetilde{G}}$ and the functional $A(f_{\widetilde{G}})$ corresponding to the genome G. Difference between programs (1), (2) for a genome and evolution is the following: for a genome (1) the genes are subject to regulation (monadic computation) while for evolution (2) possibility of application of monads is not clear (the evolution is blind).

Let us consider the evolution program $\widetilde{E}$ of the form (2) with reduction graph $\Gamma_{\widetilde{E}}(G_0)$ (constructed as in Section 2) where G_0 is the ancestor genome and vertices of the graph are descendant genomes (i.e. we generate this graph starting from the ancestor G_0).

Let us put in correspondence to action of evolution operation e_k a weight (positive number) $K(e_k)$ and to oriented path p between vertices u and v in the graph $\Gamma_{\widetilde{E}}(G_0)$ (path from ancestor to descendant) we put in correspondence the action functional given by the sum of weights of edges in the path

$$K_{\widetilde{E}}(p) = \sum_{k \in p : u \to v} K(e_{i_k}). \tag{3}$$

This functional can be considered as the cost of computation along the path p or weighted estimate for Kolmogorov complexity of generation of v from u.

Let us define Darwinian evolution as the temperature learning problem with inverse "evolution temperature" β' with statistical sum

$$Z[\widetilde{E}, G_0] = \sum_{G \in \Gamma_{\widetilde{E}}(G_0)} A(f_{\widetilde{G}}) \sum_{p \in \mathrm{Path}\left(\Gamma_{\widetilde{E}}(G_0)\right) : G_0 \to G} e^{-\beta' K_{\widetilde{E}}(p)}. \tag{4}$$

The summation runs over paths p between the ancestor genome G_0 and the descendant genome G, then we sum over descendants G.

In this formula G_0 is the ancestor genome; $G \in \Gamma_{\widetilde{E}}(G_0)$ are descendant genomes; $A(f_{\widetilde{G}})$ is the functional subject to selection (selection pressure functional); $K_{\widetilde{E}}(p)$ is the evolutionary effort to generate a descendant from the ancestor along evolution path p; summation $\sum_p$ runs over paths of evolution with the same ancestor and descendant (which describes the phenomenon of retinal evolution [7]). This is a Darwinian model of evolution by selection—the statistical sum is concentrated at genomes with large functional $A(f_{\widetilde{G}})$ (for example one could consider selection for large current functional).

Summation over paths describes parallelism in evolution (computation of typical functional $A(f_{\widetilde{G}})$ includes summation over paths which describes parallelism in metabolism). Gibbs factor $e^{-\beta' K_{\widetilde{E}}(p)}$ of the action functional reduces the complexity of evolution operations which contribute to the statistical sum of evolutionary program. This corresponds to regularization by complexity as energy and makes Darwinian evolution possible (without this regularization term we will get divergence in expression (4) for the statistical sum which corresponds to overfitting in the learning problem).

The problem of teleology ("evolution has the aim") often discussed in relation to biological evolution in this approach reduces to solvabiltity of the above learning problem. Solvability means that the aim of evolution is achieveable (i.e. the functional $A(f_{\widetilde{G}})$ can reach sufficiently large values for some G and the evolutionary effort to generate G is not very large, equivalently contribution in the summation over G in (4) is sufficiently large for some G) and there is no overfitting (i.e. the statistical sum converges).

Nondeterministic algorithm is described by a Nondeterministic Turing Machine (NTM) which at some steps of computation can duplicate and perform several branches of computation (this allows to organize brute-force search). Programs (1), (2) which describe operation and evolution of genomes are

programs for NTM since G and E are multivalued functions and recursive application of multivalued functions generate many branches of computation. This kind of parallelism is described by the syntax of applicative list functor in Haskell. We propose to consider parallelism in biology (parallelism of processes in cells and in evolution) as a manifestation of nondeterministic algorithms. Biological processes correspond to nondeterministic computations and Darwinian evolution is a temperature learning problem for a functional nondeterministic algorithm.

FUNDING

This work is supported by the Russian Science Foundation under grant 19-11-00320.

REFERENCES

1. S. V. Kozyrev, "Genome as a functional program," Lobachevskii J. Math. **41** (12), 2326–2331 (2020). arXiv: 2006.09980
2. S. V. Kozyrev, "Biology as a constructive physics," p-Adic Numbers, Ultrametr. Anal. Appl. **10**, 305–311 (2018); arXiv: 1804.10518.
3. S. V. Kozyrev, "Learning problem for functional programming and model of biological evolution," p-Adic Numbers, Ultrametr. Anal. Appl. **12**, 112–122 (2020).
4. J. Backus, "Can programming be liberated from the von Neumann style? A functional style and its algebra of programs," Comm. ACM **21**, 613–641 (1978).
5. M. Lipovaca, *Learn You a Haskell for Great Good!: A Beginner's Guide* (No Starch Press, 2011).
6. E. V. Koonin, *The Logic of Chance: The Nature and Origin of Biological Evolution* (FT Press, 2012).
7. D. H. Huson, R. Rupp, and C. Scornavacca, *Phylogenetic Networks* (Cambridge Univ. Press, Cambridge, 2010).
8. A. M. Turing, "Can machines think? Computing machinery and intelligence," Mind **49**, 433–460 (1950).
9. V. N. Vapnik, *The Nature of Statistical Learning Theory* (Springer, Berlin, 1995).
10. Y. I. Manin, "Complexity vs energy: Theory of computation and theoretical physics," J. Phys.: Conf. Ser. **532**, 012018 (2014); arXiv: 1302.6695.
11. J. Schmidhuber, "Discovering neural nets with low Kolmogorov complexity and high generalization capability," Neural Netw. **10**, 857–873 (1997).