Is genome written in Haskell?

S.V. Kozyrev, Steklov Mathematical Institute

Approach *"genome as a program"* using functional programming — genome is a functional program.

Functional programming — parallelism, simple system of states allows easy modification of programs (error control).

Comparison to biology — parallelism of processes in cells and in evolution, random modifications of genetic program in evolution usually do not break the program immediately.

Darwinian evolution — generation of programs by data
— machine learning.

Learning problems for functional programming.

Gene regulation — monadic computations
Lac operon — IO monad

Gibbs distribution and scaling: genomics, Zipf's law

Scaling in sizes of families of paralogous genes, scaling in metabolic networks and networks of interacting genes (scale free graphs).

E.V.Koonin: genome is a *"gas of interacting genes"*, scaling should be related to Gibbs distribution for this model.

Yu.I.Manin: model of statistical mechanics with Hamiltonian equal to Kolmogorov complexity (*"Complexity as Energy"*) — Gibbs distribution should give the Zipf's scaling law for distribution of words in texts.

S.K.: These two approaches can be unified if biological evolution is a model of temperature learning with regularization equal to estimate for Kolmogorov complexity.

Yu. I. Manin, Complexity vs energy: theory of computation and theoretical physics, Journal of Physics: Conference Series 532 (2014) 012018. arXiv:1302.6695

E. V. Koonin, The Logic of Chance: The Nature and Origin of Biological Evolution, FT Press, 2012.

Y.I. Wolf, M.I. Katsnelson, E.V. Koonin, Physical foundations of biological complexity, PNAS, 115:37 (2018) E8687.

S.V. Kozyrev, Biology as a constructive physics, p-Adic Numbers, Ultrametric Analysis and Applications, 10:4 (2018), 305–311. arXiv:1804.10518

S.V. Kozyrev, Learning problem for functional programming and model of biological evolution, p-Adic Numbers, Ultrametric Analysis and Applications, 12:2 (2020), 112–122.

S.V. Kozyrev, Genome as a functional program, Lobachevskii Journal of Mathematics, 41:12 (2020), 2326–2331. arXiv:2006.09980

Biology:

Molecules are linear polymers (proteins and nucleic acids) — strings of symbols, state of a system — set of strings with multiplicity (multistring). S — set of multistrings.

Chemical reactions — transformations of multistrings local in substrings (gluing, cuttings, substitutions, duplications, other multistring editing operations). Physical transformations (transfer of molecules) — changes of multiplicities of strings in multistrings.

Genome — set of genes, each gene defines a transformation of multistrings. Each gene is represented by a string, genome is a multistring.

Genes as transformations operate in parallel.

Biological evolution — transformation of genomes as multistrings by a set of genome editing operations.

Gene g_k as a function $S \to S$ is multivalued. Example: 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 to different strings in a multistring.

Genome — list $G = [g_1, \ldots, g_n]$ of genes is a multivalued function $S \to S$: any function g_k in the list can be applied to $v \in S$.

Genome as a functional program is defined recursively by genome as a list of functions $G = [g_1, \ldots, g_n]$ (genes)

$$\widetilde{G} = \widetilde{G} \circ G = [\widetilde{G} \circ g_1, \ldots, \widetilde{G} \circ g_n]. \qquad (1)$$

Functional programming

— lambda calculus, Haskell programming language

A. Church, A set of postulates for the foundation of logic, Annals of Mathematics. Series 2. 33 (2), 346–366 (1932).

J. Backus, Can Programming Be Liberated from the von Neumann Style? A Functional Style and its Algebra of Programs, Comm. ACM 21 (8), 613–641 (1978).

M. Lipovaca, Learn You a Haskell for Great Good!: A Beginner's Guide, No Starch Press, 2011.

Some notations from Haskell

Class of types Functor contains method fmap

```
fmap :: (a -> b) -> f a -> f b
```

Example: list functor

```
a->[a]
```

```
fmap f [a1,...,an] = [f a1,...,f an]
```

Applicative functors: supports operations pure and <*>.

```
class (Functor f) => Applicative f where
```

```
pure :: a -> f a
```

```
(<*>) :: f (a -> b) -> f a -> f b
```

For the list functor

```
pure :: a -> [a]
```

<*>: applications of functions from the left list to elements of the right list

```
[(*0),(+100),( 2)] <*> [1,2,3] = [0,0,0,101,102,103,1,4,9]
```

Genome as a program (1) is defined by recursive application of list of functions as applicative functor.

Monads — applicative functors supporting the bind operation

```
return :: a -> m a
```

```
(>>=) :: m a -> (a -> m b) -> m b
```

Example:

```
[1,2] >>= \x -> [x,-x] = [1,-1,2,-2]
```

Lac operon. 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.

Description of lac operon as monadic IO operation in Haskell–like syntax:

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

Functions glucose and lactose return boolean values for operations of input–output (which check presence of glucose and lactose correspondingly) and structural gene 1, 2, 3 perform operations of expression of corresponding genes.

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 (sets of molecules).

Expression of structural genes for genome as a program can be understood as action of transformations performed by structural genes (by proteins encoded by these genes).

Monadic operation `return` (put value in context) is given by binding of regulatory molecules to binding sites in corresponding operons.

Metabolic network as a reduction graph

Let $v_0 \in S$ be a multistring ("reasonable" in biological sense). Let us define a graph $\Gamma_{\widetilde{G}}$ for the program $\widetilde{G}$:

Step 0) We start from vertex v_0.

Step 1) Let us apply G to v_0, any $g_k \in G$ can be applied to v_0. 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 to obtained at the previous step vertices multivalued map G. 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). Iteration of the process gives graph $\Gamma_{\widetilde{G}}$.

Some genes g_k in G correspond to transfer operations which change multiplicities of some strings in a multistring. These operations allow to close metabolic cycles in the graph.

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 $\Gamma_{\widetilde{G}}$ — 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 $f_{\widetilde{G}}$. This distribution describes nonequilibrium thermodynamics of a biological system.

Functional $A(f)$ of biological function — some linear functional of distribution $f(v)$ on vertices of the graph. Example: 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 chemical reaction. To obtain the complete current one has to sum up values $r_+f(v_1) - r_-f(v_2)$ over all such edges. $A(f_{\widetilde{G}})$ — current in the stationary state.

Remarks

The program $\widetilde{G}$ is highly parallel. The parallelism in operation of genome can be described by the Haskell syntax of applicative list functor (list of genes — list of functions).

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 $\Gamma_{\widetilde{G}}$.

Gene regulation — 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 gene regulation works by regulatory and signal molecules which regulate expression of genes.

Analog in functional programming — monads (computations with effects, modification of states, input/output operations — interaction with the environment).

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).

Another mechanism of gene regulation: epigenetics — genome methylation, histone code, folding of chromatin.

Any gene g_k in a genome $G = [g_1, \ldots, g_n]$ is encoded by some biological sequence, i.e. gene g_k is a string and a genome G is a multistring in S.

Biological evolution — action of "evolutionary program" $\widetilde{E}$ with "evolution genes" $E = [e_1, \ldots, e_m]$ (operations of editing of genomes) defined recursively

$$\widetilde{E} = \widetilde{E} \circ E = [\widetilde{E} \circ e_1, \ldots, \widetilde{E} \circ e_m]. \qquad (2)$$

Evolution transforms genomes to genomes (as multistrings), transforms rates $r_+(g)$, $r_-(g)$ for genes, the stationary state $f_{\widetilde{G}}$ and functional $A(f_{\widetilde{G}})$ of biological function.

Difference between evolution (2) and genome (1) programs — the evolution does not support monads (analogs of gene regulation).

The evolution is blind.

Darwinian evolution — machine learning (generation of a program by data). Machine learning was proposed by A.Turing, he also mentioned analogy to Darwinian evolution.

A. M. Turing, Can machines think? Computing Machinery and Intelligence. Mind 49: 433–460 (1950).

Problem of machine learning — minimization over the space of parameters of the sum of the loss (or risk) functional and the regularization functional

$$H(s, \mathrm{data}) = R(s, \mathrm{data}) + Reg(s, \mathrm{data}) \to \min .$$

Overfitting — low value of the risk functional at a training sample and high value of risk at a control sample.
Regularization is important to control overfitting (by reducing entropy of the space of parameters s, see VC–theory).

V.N. Vapnik, The Nature of Statistical Learning Theory, Springer, 1995.

Temperature learning — 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).

Critical behavior: the functional $H(\alpha, s) = R(s) + \alpha\, Reg(s)$, $\alpha > 0$ and statistical sum $Z_\alpha = \sum_s e^{-\beta H(\alpha, s)}$.

large α — the statistical sum converges
small α — the statistical sum diverges due to summation over space of large entropy (with suitable regularization)

Let us consider divergence of Z_α as a criterion of overfitting in the learning problem — 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).

Ideas of machine learning in evolution — regularization by estimate of Kolmogorov complexity to control overfitting.
Universal scaling in genomics can be explained by universal regularization by complexity in the corresponding learning problems.

Zipf's law — scaling explained by complexity as energy model with critical temperature (Yu.I.Manin). Learning at the edge of overfitting.

Temperature learning for functional programs

Let us consider the evolution program $\widetilde{E}$ of the form (2) with genome editing operations $E = [e_1, \ldots, e_m]$ and reduction graph $\Gamma_{\widetilde{E}}(G_0)$ where G_0 is the ancestor genome (vertices are descendants).

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 — 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

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

G_0 — the ancestor genome;
$G \in \Gamma_{\widetilde{E}}(G_0)$ — descendant genomes;
$\sum_G$ — summation over descendants G;
$A(f_{\widetilde{G}})$ — functional subject to selection (selection pressure);
$K_{\widetilde{E}}(p)$ — evolution effort to generate a descendant from the ancestor along evolution path p;
$\sum_p$ — summation over paths of evolution with the same ancestor and descendant (retinal evolution).

This statistical sum is concentrated at genomes with large functional $A(f_{\widetilde{G}})$ (for example 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 term we will get divergence which corresponds to overfitting in the learning problem).

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 gives 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.
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.

Summary

Genome as a functional program (1) is defined recursively by the list of genes as applicative functor. Metabolic network is related to the reduction graph of this program. Parallelism of processes in cell and in evolution can be described by nondeterministic algorithms.

Gene regulation is described by monadic computation where monadic context is given by regulatory molecules bound to binding sites of corresponding operons. Lac operon is the IO monad.

Darwinian evolution by selection is a temperature learning problem for nondeterministic functional program (2) given by the statistical sum (4). This statistical sum contains regularization to avoid overfitting in the form of Gibbs factors of the action functional (3) of complexity as energy and describes scaling laws in genomics.

Genome can be described by a program with functional architecture written in Haskell–like language

Learning problem for functional programming is introduced