Making sense of 'genetic programs': biomolecular Post–Newell production systems

Mihnea Capraru

Abstract. The biomedical literature makes extensive use of the concept of a genetic program. So far, however, the nature of genetic programs has received no satisfactory elucidation from the standpoint of computer science. This unsettling omission has led to doubts about the very existence of genetic programs, on the grounds that gene regulatory networks lack a predetermined schedule of execution, which may seem to contradict the very idea of a program. I show, however, that we can make perfect sense of genetic programs, if only we abandon the preconception that all computers have a von Neumann architecture. Instead, genetic programs instantiate the computational architecture of Post–Newell Production Systems. That is, genetic programs are ***unordered*** sets of conditional instructions, instructions that fire independently when their conditions are matched. For illustration I present a paradigm Production System that regulates the functioning of the well-known ***lac*** operon of ***E. coli***. On close reflection it turns out that not only genes, but also proteins encode instructions. I propose, therefore, to rename genetic programs to ***biomolecular*** programs. Biomolecular and/or genetic programs, and the cellular computers than run them, are to be

understood not as von Neumann computers, but as Post–Newell production systems.

1 Background

In the biomedical literature we often encounter the concept of a genetic program. We read, for instance, that "dynamic and diverse genetic programs … assemble the human central nervous system … during development and maintain its function throughout life" (Cherry et al. 2020); "YTHDF2 suppresses the plasmablast genetic program" (2022); "microglia express a conserved core gene program of orthologous genes from rodents to humans" (Geirsdottir et al. 2019); "[t]ranscriptome analysis of mouse and human sinoatrial node cells" also "reveals a conserved genetic program" (Van Eif et al. 2019); there is such a thing as "[t]he genetic program of hematopoietic stem cells" (2000); there is an "altered genetic program in senescent human fibroblasts" (1990), and also a "genetic program for wound healing in human skin fibroblasts" (2001). On an interesting side note, genetic programs these days are not only naturally evolved, but also created—or as it were written—intentionally: thus we can read about the "[g]enetic programming of macrophages to perform anti-tumor functions using targeted mRNA nanocarriers" (Zhang et al. 2019). Thinking of genomes as programs can be enticing, because we know, or we think we know, what a computer program is. But we do not yet know, as I will argue, what a genetic program is. This is not because nothing responds to the concept, but rather because we have not been thinking clearly enough about the question. In this article I will attempt to rectify this issue and and to illuminate the nature of genetic programs.

According to a simplistic and unrealistic picture, one which we may label 'programmatic determinism', a genetic program is simply a list of step-by-step instructions for making an organism. Programmatic determinism is perhaps correctly attributed to conceptual pioneers Jacques Monod and François Jacob. The latter in particular says:

> What are transmitted from generation to generation are the 'instructions' spec-

> ifying the molecular structures: the architectural plans of the future organism. (Jacob 2022[1970])

Notice the casual transition from molecules to organisms. Few, of course, would in fact agree that a genetic program specifies an architectural plan for the whole organism. As Richard Dawkins memorably put it, genomes are not blueprints but recipes; they do not specify the finished organism, but rather the steps of its development (1986:294–96).

Even the recipe metaphor, however, does not do full justice to the functional complexity of genetic programs. One and the same cake recipe results in one and the same cake, but one and the same genetic recipe results in different cells in different tissues.[1] As Dawkins, of course, knew quite well, the same gene does not do the same thing in every context. What the gene does depends on the place, time, and circumstance at which it is switched on. In other words, genes are best seen as specifying ***conditional*** instructions. Such an instruction first evaluates a condition, a condition that involves the combined presence or absence of the protein types known as transcription factors. Then, if the condition matches, the instruction performs a certain action. The action, in turn, may result in the production of mRNA and proteins, and the latter may themselves function as transcription factors that trigger further conditional instructions.

This is why Eric Davidson has argued that 'the regulatory genome' "can be symbolized, as in a computer program, by a series of conditional logic statements" (2006:54), as he exemplifies:

if Runt and not (Myb or Z12)	i1 = Runt(t)
else	i1 = 0
if Oct	i2 = i1
else	i2 = 0
if P1 and not P3A2	i3 = P1(t)
else	i3 = 0

[1] Goldschmidt 1927.

if CBF i4 = 2◇i3

else i4 = i3

i5=i2+i4

if TEF-1 i6 = 2◇i5

else i6 = i5

(Davidson 2006:ED, p. 56)

In Davidson's vision, the genetic program is thus a sequence of conditional instructions, to which transcription factors function as inputs. A similar approach is proposed by Gary Marcus (2004:60–66), who sees genetic programs as lists of conditional instructions executed "not by a central processor but autonomously, by individual genes in individual cells" (2004:61).

If there are such things as genetic programs at all, then no doubt Davidson and Marcus are moving in the right direction, because such programs must be sensitive to transcription factors, and this sensitivity must be formalized by conditional instructions. Their approach, however, is thus far incomplete, in a manner that opens it up to a significant objection. It is this objection that will point our way forward.

2 Production systems and the missing schedule of execution

In 2014 Ronald Planer argues against the very existence of genetic programs, on the grounds that "there is no ***order*** in which these instructions can be properly said to be retrieved and executed" (2014:39). Rather, says Planer, "the cell must simply be understood as retrieving

and executing every one of these instructions simultaneously or in parallel" (2014:39). According to Planer, "[i]n a computer, the instructions in a program are contained in a memory stack and are read by the central processor in a serial manner" (2014:39). As Planer correctly points out, multi-threaded computing also executes instructions after a serial manner. Hence we cannot simply reply to Planer that genetic programs are executed in parallel. More needs to be said. As I contend, however, Planer's objection is mistaken, and rests on an incomplete understanding of computational architectures. Furthermore, once we understand why Planer is mistaken, we will also understand the correct architecture of genetic programs.

Planer's mistake is this: He assumes as obvious the premise that all computer programs are executed under the control of a central processing unit (CPU) according to a predetermined ***schedule***. In other words, he assumes that all computation proceeds according to the von Neumann architecture. This assumption does correspond to the way that most of us instinctively think about computer programs—the assumption, after all, is correct about the kind of programs we usually write in C++, Java, Python, etc. But the assumption is not correct in general. As we will see, computation can proceed according to non-von-Neumann architectures without predetermined schedules of execution. Hence what Planer's argumentation tells us is ***not*** that genetic programs do not exist, but rather, that ***if*** genetic programs exist ***then*** they are non-von-Neumann.

If not von Neumann, though, then what? Let us examine two possibilities. One was raised by Roger Sansom in his excellent book of 2011. The other will constitute my proposal in this article.

Sansom, to be clear, does not frame his argument in terms of genetic programs, but in terms of gene regulatory networks (GRNs). A GRN is a set of genes some of which regulate the rest, as well as each other, by producing transcription factors. What Sansom is concerned with is to model the evolution of GRNs, and more importantly to prove their gradual, step-by-step evolvability. To this end he argues that GRNs have the computational structure of

artificial neural networks (ANNs). Genes receive inputs from other genes in the same way in which ANN neurons receive inputs from other neurons. Furthermore, Sansom argues that GRNs ***evolve*** in the same way in which ANNs are trained. Sansom's objective is salutary, and his proof of evolvability seems convincing. But while it is important to have evolvability proofs, and to this end it may be useful to model GRNs as ANNs, ANN architecture does not fully reflect the structure of genetic programs.

Recall that genetic programs are best seen as collections of conditional instructions. In the simplest case, such an instruction is encoded by a set of cis-regulatory elements (CREs) together with a coding region. CREs are genomic sites at which transcription factors may bind, or fail to.[2] CREs encode the condition of the conditional instruction, while the coding region encodes the action:

IF (the CREs's condition is satisfied) THEN (produce mRNA)

The CREs's condition, in turn, is best seen as a logical complex built up with the truth-conditional functors of propositional logic, e. g., ***and***, ***or***, ***not***, ***xor***:

IF (transcription-factor$_1$ AND NOT transcription-factor$_2$) THEN …

IF (transcription-factor$_3$ OR transcription-factor$_4$) THEN …

Transcription factors, as we see, function as variables whose logical combinations control the computer's behavior. It is precisely this kind of logical–symbolic computation that is ***prima facie*** at odds with ANN computation. Whereas symbolic computation manipulates strings of discrete, localized symbolic tokens, ANNs compute matrix products that propagate activation holistically, from all neurons in one layer to each neuron in the next. It is possible, to be sure, to use an ANN as the base level on which to implement a higher-level symbolic machine. But

[2] Let us note that cis-regulatory elements are not the only genes that encode instructions in biomolecular programs. CREs are located next to the structural genes that they regulate. Conditional instructions, however, may also be encoded by genes located in trans, i. e., relatively far from what they regulate. A classic example is transvection (Lewis 1954). Transvection occurs in diploid organisms in which a gene A is regulated by another gene B, with B located on the chromosome homologous to A's.

in this case it is the latter, higher-level machine that performs the symbolic computations, while the ANN becomes an optional, redundant, and over-engineered implementation detail.

This, then, is why genetic programs cannot be understood as ANNs: genetic programs are ***logical*** or symbolic, in that they execute instructions based on propositional-logical combinations of transcription factors. ANNs are not logical in that way; at best, they can be used to implement symbolic computers on top of them. Hence although it is possible that GRN ***evolution*** is well analogized to ANN training, genetic programs must instantiate a different computational architecture.

This computational architecture, I maintain, is that of the Post–Newell production system (Post 1943; Newell and Simon 1972). Such systems were introduced by Emil Post to study the foundations of mathematics. Newell and Simon wanted to create human-like AI. We will use them to understand the structure of genetic programs.

A Post–Newell production system is an unordered set of mutually independent conditional instructions, of the form 'IF ***pattern*** THEN ***action***.' All of these instructions read and write to a common memory; furthermore, all instructions constantly scan the memory for matches to their patterns. Once the patterns match, production systems differ in whether they execute the actions serially or in parallel. When the actions are executed serially, it is possible for the programmer to assign priorities to specific instructions. In the case of the genome, however, the actions are executed not serially but in parallel.

It is of paramount importance that these instructions are ***unordered***. Even if the programmer types the instructions in the form of a list of IF–THEN sentences, we can shuffle that list in any arbitrary order, and it will still constitute ***exactly the same program***. In the next section I will exemplify a genetic production system. I will introduce the instructions one by one, in the order in which they are best explained. But I could have introduced them in any other order, at a cost only to the ease of explanation.

Recall, now, that Planer argues that genomes cannot encode programs, because such

programs would lack a schedule of execution. As I have explained, however, a schedule of execution is not necessary for something to constitute a program. There is also the kind of program that consists precisely of conditional instructions that are executed ***without*** a schedule: to wit, a parallel production system. This, then, is the computational architecture of genetic programs.

3 A paradigm genetic/biomolecular program: the *lac* operon

So far we have seen that if there are such things as genetic programs, then they are best regarded as Post–Newell production systems. Let us now proceed to describe a paradigm production system for the E. coli ***lac*** operon, itself a paradigm gene regulatory network.[3]

The prokaryote E. coli can metabolize both glucose and lactose, but it prefers the former. This preference manifests in the activity of the ***lac*** operon, which is customarily inactive, but activates when the cell needs to metabolize lactose. When the operon activates, it produces a single mRNA, which in turn is translated into three proteins frequently referred to as LacA, LacY (beta-galactoside permease),[4] and LacZ (or β-galactosidase). LacY has the function to allow extra-cellular lactose to cross into the cell through the cellular membrane. LacZ splits or cleaves lactose into glucose and galactose. Hence ***ceteris paribus*** more ***lac*** activity leads to more lactose, glucose, and galactose within the cell.

For the ***lac*** operon to activate, two conditions must be met: 1) the intracellular glucose level must be low, and 2) the cell must contain some amount of lactose.[5] For one thing, if there is sufficient glucose, then the cell feeds on glucose, and it doesn't matter whether lactose is available or not. For another thing, even when glucose is absent, if no lactose is present

[3]Monod 1942; Monod and Cohn 1978[1952]; Jacob, Perrin, Sánchez, Monod, et al. 1960; Jacob and Monod 1961a; Jacob and Monod 1961b.
[4]Abramson et al. 2003.
[5]Monod 1942; Novick and M. Weiner 1957.

either, then making LacY and LacZ would be a waste, so the operon does not activate. Only when glucose is insufficient, but lactose is available, is the operon activated.

These rules are embodied in a conditional of the type we have discussed:

(1) IF there is no glucose but there is lactose THEN activate the ***lac*** operon

Let us pursue the details. The relevant Boolean variables are implemented by the transcription factors cAMP–CRP and LacI. The cAMP–CRP complex[6] is produced when glucose is low. This complex is a ***lac*** activator; in its absence, ***lac*** remains inactive. LacI[7] is a repressor; it is usually bound to one or to several operator sites, so that RNA polymerase cannot bind to the operon's promoter. When lactose is absorbed into the cell, a small amount of its isomer allolactose results. Allolactose binds to LacI and disables it either partially or completely. When LacI is thus disabled, RNA polymerase can bind to the promoter. More formally:

(2) IF is-bound(cAMP-CRP) AND NOT is-bound(LacI)
THEN IF (rand() $\leq p_{lac}$)
THEN **bind** RNApolymerase **to** ***lac***

(3) IF is-bound(RNApolymerase, ***lac***)
THEN **produce**(LacA) AND **produce**(LacY) AND **produce**(LacZ)

Let us explain the above. Logical constants are typeset in capitals: 'IF', 'AND', etc. Actions are bold-faced. rand() is a function that returns a random real number between 0 and 1, while p_{lac} is the probability that RNA polymerase will attach to ***lac*** when cAMP-CRP is present and LacI absent. In English: if the relevant attachment sites contain cAMP-CRP but not LacI, then bind RNA polymerase to ***lac***; but do so only a certain percentage of the time (p_{lac}). Furthermore, when RNA polymerase binds to ***lac***, produce the three proteins LacA, LacY, and LacZ.

[6]Perlman, Crombrugghe, and Pastan 1969; Beckwith, Grodzicker, and Arditti 1972; McKay and Steitz 1981; Aiba, Fujimoto, and Ozaki 1982; Cossart and Gicquel-Sanzey 1982; Schultz, Shields, and Steitz 1991.
[7]Gilbert and Müller-Hill 1966; Beyreuther, Adler, Geisler, and Klemm 1973; Farabaugh 1978.

Let us take a more detailed look at the operation of LacI. The ***lac*** operon is controlled by three distinct operators, O1,[8] O2, and O3.[9] When LacI binds only to O1, it reduces ***lac*** activity by 95%. Thus the operon is still somewhat active, and the cell retains a limited ability to metabolize lactose. But when LacI binds to O1 and also to O2 or to O3, then the operon loops and its activity is reduced by more than 99.9%.[10]

Therefore to the instruction (2) we will add (4), (5), and (6):

(4) IF is-bound(cAMP–CRP) AND is-bound(LacI, O1)
AND NOT is-bound(LacI, O2) AND NOT is-bound(LacI, O3)
THEN IF (rand() ≤ 0.95)
THEN **block** RNApolymerase **at** ***lac***
ELSE IF (rand() $\leq p_{lac}$)
THEN **bind** RNApolymerase **to** ***lac***

(5) IF is-bound(cAMP–CRP) AND is-bound(LacI, O1) AND is-bound(LacI, O2)
AND NOT is-bound(LacI, O3)
THEN IF (rand() ≤ 0.999986)
THEN **block** RNApolymerase **at** ***lac***
ELSE IF (rand() $\leq p_{lac}$)
THEN **bind** RNApolymerase **to** ***lac***

(6) IF is-bound(cAMP–CRP) AND is-bound(LacI, O1) AND is-bound(LacI, O3)
AND NOT is-bound(LacI, O2)
THEN IF (rand() ≤ 0.999978)
THEN **block** RNApolymerase **at** ***lac***
ELSE IF (rand() $\leq p_{lac}$)
THEN **bind** RNApolymerase **to** ***lac***

[8] Gilbert and Maxam 1973.
[9] Reznikoff, Winter, and Hurley 1974.
[10] Oehler, Eismann, Krämer, and Müller-Hill 1990; Santillán and Mackey 2008; Narang 2007.

In English, here is what the instructions above are saying: If there is too little glucose (if cAMP–CRP is bound), and if the repressor LacI is bound only to O1, ***resp.*** both to O1 and O2, ***resp.*** both to O1 and O3, then block RNA polymerase 95% of the time, ***resp.*** 99.9986% of the time, ***resp.*** 99.9978% of the time. Otherwise, bind RNA polymerase $p_{lac} \times 100\%$ of the time.

Our instructions so far tell the cellular computer what to do depending on whether LacI is bound to nothing, or to O1 alone, or to O1 as well as to one of the other operators O2 and O3. What, though, determines whether LacI is likely to do one of these or another? It is the presence and concentration of allolactose, which furnishes information on whether lactose is available.

Interestingly, LacI and ***lac*** do not work on a simple on/off principle; rather, ***lac*** is best described as having three modes: on, mostly-off, and almost-fully-off. This allows the cell to modulate ***lac*** activity depending on the allolactose concentration: the more allolactose, the more LacY and LacZ are worth producing.

Let us now ask: Which part of the computer is it that performs this adjustment? It is not the genes themselves that do so. Instead, it is LacI, a protein. This brings us to a somewhat surprising conclusion: in order to make sense of 'genetic programs', we should perhaps replace this very expression with a more extensive or inclusive one, to wit, ***biomolecular programs***.

Consider for illustration the following two instructions:

(7) IF neighbors(LacI, allolactose)
THEN **bind**(LacI, allolactose) **into** LacI+$_1$allolactose

(8) IF neighbors(LacI+$_1$allolactose, allolactose)
THEN **bind**(LacI+$_1$allolactose, allolactose) **into** LacI+$_2$allolactose

According to instruction (7), if a LacI molecule meets ***one*** allolactose molecule, they will bind into a complex we may label LacI+$_1$allolactose. This complex can no longer bind two

operators at once, but it can still bind O1. Recall that if only O1 is bound, then ***lac*** activity is only reduced by 95%; it is only when one of O2 and O3 is also bound, that ***lac*** turns off almost completely.

According to instruction (8), if the LacI+$_1$allolactose complex meets yet another allolactose molecule, it now binds the second allolactose and becomes what we have labeled LacI+$_2$allolactose. This new compound can no longer bind even one single operator. ***lac*** is therefore completely active:[11]

(9) IF neighbors(LacI, O1, O2) THEN **bind**(LacI, O1, O2)

(10) IF neighbors(LacI, O1, O3) THEN **bind**(LacI, O1, O3)

(11) IF neighbors(LacI, O1) THEN **bind**(LacI, O1)

(12) IF neighbors(LacI+$_1$allolactose, O1) THEN **bind**(LacI+$_1$allolactose, O1)

(13) IF neighbors(LacI+$_2$allolactose, O1) THEN **continue**

Likewise for the activator cAMP–CRP:

(14) IF neighbors(CRP, cAMP) THEN **bind**(CRP, cAMP) **into** (cAMP–CRP)

(15) IF neighbors(cAMP–CRP, ***lac***-promoter) THEN **bind**(cAMP–CRP, ***lac***-promoter)

4 Further elucidation

Let us take a more explicit look at the way in which the ***lac*** operon instantiates the features of a production system. First, recall that in a production system, all the conditional instructions access the same common memory store, which they search for matches to their

[11] At the level of chemical hardware, the implementation is slightly more complex than our functional software description above. LacI is a tetramer that can bind allolactose at each of its four constituent parts. If one ***or*** two allolactose molecules bind ***on the same side*** of the tetramer (on the same dimer), what results is functionally what we have labeled LacI+$_1$allolactose. If two allolactose molecules bind on opposite sides of the tetramer, then we obtain LacI+$_2$allolactose.

conditions. Within the cell, the memory store is implemented by the chemical landscapes of the cytoplasm for prokaryotes, the nucleoplasm for eukaryotes, or once again the cytoplasm for prokaryote-like organelles such as mitochondria and chloroplasts. Memory searching is implemented in multiple ways. The simplest search consists of transcription factors diffusing three-dimensionally through the intracellular soup. But transcription factors often bind much faster than 3D diffusion would predict. This is because of additional search mechanisms, of which one involves transcription factors sliding one-dimensionally down the DNA, until they meet a gene they can bind to (Elf, Li, and Xie 2007).

As I emphasized, in a production system the conditional instructions are independent of each other, in the sense that it only matters what the instructions say, but not in which order they say it. For instance, (16) and (17) are two ways to write one and the same program:

(16) (a) IF condition$_1$ THEN perform action$_1$

(b) IF condition$_2$ THEN perform action$_2$

(17) (a) IF condition$_2$ THEN perform action$_2$

(b) IF condition$_1$ THEN perform action$_1$

In a cellular computer, the conditional instructions are also independent of each other. Each instruction is triggered as soon as its condition matches, with no regard to whatever may be going on elsewhere in the genome. There is no such thing in the cell as 'the next instruction,' and no instruction needs to wait its time before another one has completed. When an instruction is triggered, this is not because it comes next in some scheduled order: it could just as well have been triggered sooner, if only the right transcription factors had been present or absent at the right places.

I have also mentioned that a parallel production system is disanalogous from a multi-threaded algorithm. But a multi-threaded algorithm is also executed in parallel. What, then, is the disanalogy? While a multi-threaded algorithm does run on parallel threads, each such

thread has its own schedule of execution. A production system, on the other hand, does not have parallel schedules of execution: it has no schedules at all. Hence the disanalogy between production systems and multi-threaded algorithms is the same as that between zero schedules and multiple schedules.

In an irenic spirit, production systems may perhaps be conceptualized as trivially multi-threaded, in the sense that each conditional instruction constitutes its own thread, on which it ceaselessly scans for matches to its own condition.[12] Perhaps; but a schedule with only one entry is a schedule in name only. When Planer points out that genetic programs lack a schedule of execution, he is implicitly referring to schedules that contain at least two entries, of which one is executed simply because it is next, and because the other entry has completed. This nontrivial kind of schedule is what von Neumann computation executes, whether it is single-threaded or not. By contrast, in a cellular production system, instructions don't wait their turn.

Finally, programs running on a computer often engage in so-called inter-process communication. This can take sundry forms, from ordinary files to locks, sockets, pipes, etc. Interestingly, biomolecular programs possess the same ability, in the guise of cell signaling. Conditional instructions in one cell can produce hormones, pheromones, or neurotransmitters that activate further instructions in another cell. This plays a crucial coordinating role during development, and helps to solve Goldschmidt's problem of how one and the same genome can produce all the different cells at all the right places within an organism.

5 Conclusion

Numerous cellular processes function according to computer-like programs. Such programs are instantiated by basic building blocks as old as life itself. Unlike regular computer pro-

[12] It is also true that if we want to implement a production-system virtual machine on top of an ordinary von Neumann computer, then it is an excellent idea to make the virtual machine multi-threaded. But this is an implementation detail.

grams, which are written by intelligent programmers, cellular programs result from the same kind of undirected self-organization and natural selection as other biological adaptions. (An exception, perhaps, is provided by certain products of genetic engineering.)

As we see in our paradigm description of a production system for the ***lac*** operon, it is not only genes, but also proteins such as LacI that sense and process information within the cell. I reiterate therefore my proposal to relabel genetic programs as biomolecular ones. At the same time, I admit that one may prefer the old label, not because it is more descriptive, but because it is more easily recognized.

Be that as it may, I have argued that genetic or biomolecular programs are disanalogous from the kind of programming that is most frequently taught in schools and used in real-world technological applications. Since our programs lack a predetermined schedule of execution, they do not instantiate the von Neumann computational architecture. Rather, they have turned out to be Post–Newell production systems. This latter architecture has been created originally not to investigate genetic programs, but rather to model human cognition and to create human-like AI. As it sometimes happens, however, fertile cross-polination between fields is now possible.

References

Abramson, Jeff, Irina Smirnova, Vladimir Kasho, Gillian Verner, H Ronald Kaback, and So Iwata (2003). "Structure and mechanism of the lactose permease of Escherichia coli." In: ***Science*** 301, pp. 610–615.

Aiba, Hiroji, Shinji Fujimoto, and Norihito Ozaki (1982). "Molecular cloning and nucleotide sequencing of the gene for E. coli cAMP receptor protein." In: ***Nucleic Acids Research*** 10, pp. 1345–1361.

Beckwith, Jon, Terri Grodzicker, and Rita Arditti (1972). "Evidence for two sites in the lac promoter region." In: ***Journal of molecular biology*** 69, pp. 155–160.

Beyreuther, Konrad, Klaus Adler, Norbert Geisler, and Alex Klemm (1973). "The amino-acid sequence of lac repressor." In: ***Proceedings of the National Academy of Sciences*** 70, pp. 3576–3580.

Chen, Chih-Chiun, Fan-E Mo, and Lester F. Lau (2001). "The Angiogenic Factor Cyr61 Activates a Genetic Program for Wound Healing in Human Skin Fibroblasts*." In: ***Journal of Biological Chemistry*** 276, pp. 47329–47337.

Cherry, Timothy J, Marty G Yang, David A Harmin, Peter Tao, Andrew E Timms, Miriam Bauwens, Rando Allikmets, Evan M Jones, Rui Chen, Elfride De Baere, et al. (2020). "Mapping the cis-regulatory architecture of the human retina reveals noncoding genetic variation in disease." In: ***Proceedings of the National Academy of Sciences*** 117, pp. 9001–9012.

Cossart, Pascale and Brigitte Gicquel-Sanzey (1982). "Cloning and sequence of the crp gene of Escherichia coli K 12." In: ***Nucleic acids research*** 10, pp. 1363–1378.

Davidson, Eric H. (2006). ***The regulatory genome: gene regulatory networks in development and evolution***. Academic Press.

Dawkins, Richard (1986). ***The blind watchmaker: why the evidence of evolution reveals a universe without design***. W. W. Norton & Company.

Elf, Johan, Gene-Wei Li, and X Sunney Xie (2007). "Probing transcription factor dynamics at the single-molecule level in a living cell." In: ***Science*** 316, pp. 1191–1194.

Farabaugh, Philip J (1978). "Sequence of the lacI gene." In: ***Nature*** 274, pp. 765–767.

Geirsdottir, Laufey, Eyal David, Hadas Keren-Shaul, Assaf Weiner, Stefan Cornelius Bohlen, Jana Neuber, Adam Balic, Amir Giladi, Fadi Sheban, Charles-Antoine Dutertre, et al. (2019). "Cross-species single-cell analysis reveals divergence of the primate microglia program." In: ***Cell*** 179, pp. 1609–1622.

Gilbert, Walter and Allan Maxam (1973). "The nucleotide sequence of the lac operator." In: ***Proceedings of the National Academy of Sciences*** 70, pp. 3581–3584.

Gilbert, Walter and Benno Müller-Hill (1966). "Isolation of the lac repressor." In: ***Proceedings of the National Academy of Sciences*** 56, pp. 1891–1898.

Goldschmidt, R. (1927). ***Physiologische theorie der vererbung***. J. Springer.

Grenov, Amalie, Hadas Hezroni, Lior Lasman, Jacob H. Hanna, and Ziv Shulman (2022). "YTHDF2 suppresses the plasmablast genetic program and promotes germinal center formation." In: ***Cell Reports*** 39.

Jacob, François (2022[1970]). ***The logic of life: a history of heredity***. Princeton University Press.

Jacob, François and Jacques Monod (1961a). "Genetic regulatory mechanisms in the synthesis of proteins." In: ***Journal of Molecular Biology*** 3.3, pp. 318–356.

— (1961b). "On the regulation of gene activity." In: ***Cold Spring Harbor symposia on quantitative biology***. Vol. 26. Cold Spring Harbor Laboratory Press, pp. 193–211.

Jacob, François, David Perrin, Carmen Sánchez, Jacques Monod, et al. (1960). "The operon: a group of genes whose expression is co-ordinated by an operator." In: ***Compte Rendu de l'Academie des Sciences*** 250, pp. 1727–1729.

Marcus, Gary (2004). ***The birth of the mind: how a tiny number of genes creates the complexities of human thought***. Basic Books.

McKay, David B and Thomas A Steitz (1981). "Structure of catabolite gene activator protein at 2.9 Å resolution suggests binding to left-handed B-DNA." In: ***Nature*** 290, pp. 744–749.

Monod, Jacques (1942). "Recherches Sur La Croissance Des Cultures Bactériennes." In.

Monod, Jacques and Melvin Cohn (1978[1952]). "La biosynthèse induite des enzymes (adaptation enzymatique)." In: ***Selected Papers in Molecular Biology by Jacques Monod***. Academic Press, pp. 220–272.

Narang, Atul (2007). "Effect of DNA looping on the induction kinetics of the lac operon." In: ***Journal of theoretical biology*** 247, pp. 695–712.

Newell, A. and H.A. Simon (1972). ***Human Problem Solving***. Prentice-Hall.

Novick, Aaron and Milton Weiner (1957). "Enzyme induction as an all-or-none phenomenon." In: ***Proceedings of the National Academy of Sciences*** 43, pp. 553–566.

Oehler, Stefan, Elisabeth R Eismann, Helmut Krämer, and Benno Müller-Hill (1990). "The three operators of the lac operon cooperate in repression." In: ***The EMBO journal*** 9, pp. 973–979.

Perlman, Robert L, Benoit de Crombrugghe, and Ira Pastan (1969). "Cyclic AMP regulates catabolite and transient repression in E. coli." In: ***Nature*** 223, pp. 810–812.

Phillips, R. L., R. E. Ernst, Ivanova Brunk B., M. A. N. Mahan, J. K. Deanehan, K. A. Moore, G. C. Overton, and I. R. Lemischka (2000). "The genetic program of hematopoietic stem cells." In: 288, pp. 1635–40.

Planer, Ronald J. (2014). "Replacement of the 'Genetic Program' Program." In: ***Biology and Philosophy*** 29, pp. 33–53.

Post, Emil (1943). "Formal reductions of the general combinatorial decision problem." In: ***American Journal of Mathematics*** 65, pp. 197–215.

Reznikoff, William S, Robert B Winter, and Carolyn Katovich Hurley (1974). "The location of the repressor binding sites in the lac operon." In: ***Proceedings of the National Academy of Sciences*** 71, pp. 2314–2318.

Sansom, Roger (2011). "Ingenious genes: how gene regulation networks evolve to control development." In.

Santillán, Moisés and Michael C Mackey (2008). "Quantitative approaches to the study of bistability in the lac operon of ***Escherichia coli***." In: ***Journal of the Royal Society Interface*** 5, pp. 29–39.

Schultz, Steve C, George C Shields, and Thomas A Steitz (1991). "Crystal structure of a CAP-DNA complex: the DNA is bent by 90." In: ***Science*** 253, pp. 1001–1007.

Seshadri, Tara and Judith Campisi (1990). "Repression of c-fos Transcription and an Altered Genetic Program in Senescent Human Fibroblasts." In: ***Science*** 247, pp. 205–209.

Van Eif, Vincent WW, Sonia Stefanovic, Karel Van Duijvenboden, Martijn Bakker, Vincent Wakker, Corrie De Gier-de Vries, Stéphane Zaffran, Arie O Verkerk, Bas J Boukens, and Vincent M Christoffels (2019). "Transcriptome analysis of mouse and human sinoatrial node cells reveals a conserved genetic program." In: ***Development*** 146, dev173161.

Zhang, Fan, NN Parayath, CI Ene, SB Stephan, AL Koehne, ME Coon, EC Holland, and MT Stephan (2019). "Genetic programming of macrophages to perform anti-tumor functions using targeted mRNA nanocarriers." In: ***Nature communications*** 10, p. 3974.