Cells as Information Processors

Part I: Formal Software Principles

Royal Truman*

Abstract

Cells perform millions of Boolean logic operations every second using multiple independent codes with stringent formal rules instantiated on DNA, RNA, proteins, sugars, and membranes. These codes rely on elementary and concatenated symbols to define variables and values that can be written, deleted, and read from long- and short-term memory. Computer and cellular variables are used with control structures such as "GoTo," subroutine calls, "wait," and to initiate and terminate iteration loops. They have well-defined data types and allowed operations. Values can be structured in arrays and linked lists.

Although variables are identifiable in cells, logic is executed without a readable source code, using hardwired biochemical components and inherited molecular machines (MMs). Each code requires unique decoding MMs, and cellular codes interoperate to incorporate details located throughout the cell to permit holistic correct decisions. Tight integration between these codes is implemented using adaptor biomolecules. DNA, RNA, and proteins are used to define both variables and values for independent codes, often in overlapping regions. These biomolecules are also needed to create MMs, adaptors, and the rest of the infrastructure.

Introduction

Biological research and interpretation have been dominated by philosophical naturalism for almost two centuries, especially when considering the question of origins. Deliberate design is often rejected as unscientific, which leads to even absurd proposals being entertained since "something must have happened." This is remarkable, since we interact daily with a world affected by conscious decision making. If we found a computerlike object on Mars, most would not insist on finding an explanation limited to deep time, random mutations, natural selection, chemistry, and physics. Although it would be possible to also explain the actions of a chess-playing program post-facto by tracing a series of internal mechanistic steps, this explanation would be incomplete. It would fail to explain the innate ability to anticipate and solve novel complex problems.

Prokaryote and eukaryote cells can do far more than a chess-playing program, being able to solve an aston-

* Dr. Royal Truman, Mannheim, Germany, royaltruman@yahoo.com

Accepted for publication April 22, 2016

ishing variety of unrelated problems concurrently. A seemingly endless list of contingencies has been anticipated, even when the exact details were never encountered before by the cell or its ancestors. Flexible categories of problems have been foreseen. Cells perform logic processing in a manner surprisingly similar to computers, using codes, structured datatypes, variables, algorithmic constructs such as Boolean logic and iteration, and a hierarchy of sophisticated data storage strategies for short- and long-term memory. Ignoring this integrated, holistic aspect of cells and insisting on a reductionist neo-Darwinian explanation for every cellular feature prevents answering the relevant questions correctly: Where did they come from, and why are they there?

Interpreting Biological Change and Development

Many complex processes exhibited by living systems suggest an intention or purpose. Examples include migration of birds to specific locations during certain time periods, development of adults from a fertilized cell, metamorphosis of caterpillars into butterflies, and execution of a strategy based on mental processes. This led philosophers long ago to embed purpose in physical objects as a form of internal will. Aristotle identified four kinds of causes for movement and change in general—the material, formal, efficient, and final—and claimed in Book II of *Physics* that a stone falls because it has an internal nature that drives it to attain its natural state. Many prominent thinkers since then have tried to interpret the specialness of living systems using notions such as a "formative drive," "living principle," "life-energy," "entelechy," and "teleology."

Currently, however, science has become dominated by reductionist and mechanistic thinking typified by books such as Jacques Loeb's *The Mechanistic Conception of Life* published in 1912 and the works of behaviorist psychologists—in particular B. F. Skinner—who deny the existence of will and mental states that perceive and direct behavior. This misguided naturalist thinking distorts much of what we observe and experience. Purpose and guidance are apparent and need to be taken into account. The existence and operation of an orchestra, growth of trees, poker-playing programs, and so on cannot be adequately explained by extrapolation from the natural behavior of many atoms. Wilhelm Dilthey (1833–1911), prominent philosophy professor at the University of Berlin, had a special interested in scientific methodology and introduced a distinction between the humanities (*Geisteswissenschaften*) and natural sciences (*Naturwissenschaften*). He argued correctly that investigative methods are often being applied in areas they are unsuitable for.

Purpose and guidance in nature need to be revisited. In this two-part series, we will examine how intent is governed in cellular processes, using digital computers as a model. We will show formal software principles are involved, which are processed by hardware molecular machines (Scruton, 1996, p. 254). University of Chicago microbiology professor James Shapiro referred to such stored instructions in a recent lecture, pointing out, "Cells use cognitive processes (=action based on knowledge) in dealing with genomic information" (Shapiro, 2011). At the conclusion of this analysis, we are reminded of Aristotle's claim that we cannot understand any cause for change until we can deduce its purpose (Stangroom and Garvey, 2005, p. 17).

Examples of Complex Programs in Cells

Prokaryote and eukaryote cells contain hundreds of integrated and carefully regulated programs such as metabolic networks and signal cascades linking the environment with gene regulation. Complex multicellular organisms display gene regulatory networks to unfold developmental programs and generate nervous systems and brain microcircuitries (Markram et al., 2015). We will examine these and other examples below and in the next paper. In all cases well-defined, logic-processing steps are involved, which channel the outcomes.

Coded Information Systems

In a series of papers, Truman introduced the theory of coded information systems (CISs), a framework to interpret how information-driven systems work. A CIS consists of linked tools or machines that refine outcomes to attain a specific goal (Truman, 2012a, 2012b, 2012c, 2013, 2015) (Figure 1). A coded message must play a prominent role between at least two members of these linked processes to demarcate from simple machines. Messages satisfy rules and strict formalisms to be interpreted reliably and provide flexibility and multipurposes (Hofstadter, 1980, p. 26).

Intended outcomes are ensured in a CIS through refinements using a combination of four possible "refinement factors": coded messages, sensors, physical hardware, and preexisting resources such as data or logic-processing algorithms. The model is quantitative, measuring the decreased entropy with respect to a reference state between each refinement step.

Often the CIS first increases the range of possible outcomes before applying constraining processes. To illustrate, the coding portion of a particular gene specifies a subset of useful protein sequences. How has entropy been decreased? The reference entropy to compare against is the variety of polypeptides that could be generated thanks to the cellular environment (without DNA, RNA polymerase, ribosome, ATP, tRNAs, and other resources, these long linear chains do not form naturally). The

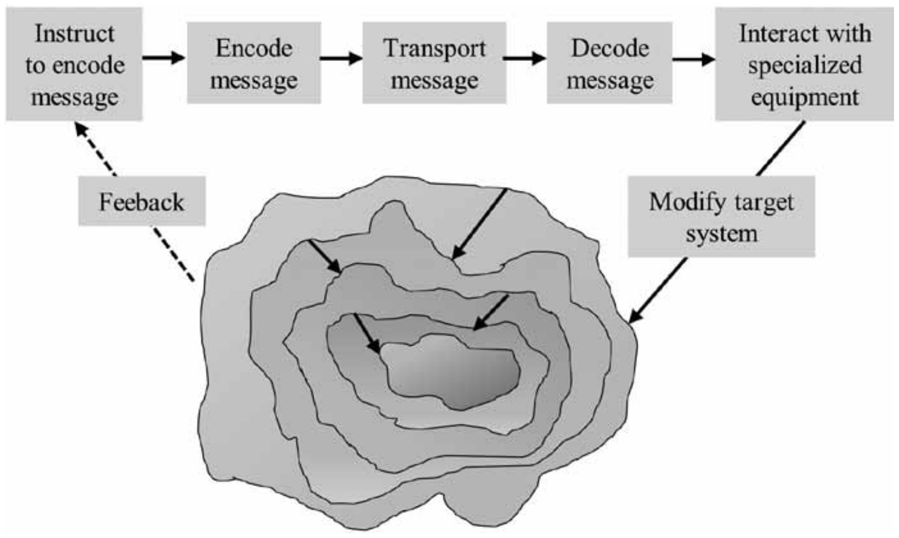


Figure 1. Coded Information Systems sequentially refine behavior through a series of processes. Each goal-directing refinement step could be influenced through coded messages, sensors, physical hardware, or preexisting resources such as data or logic-processing algorithms. At least one process must be guided by coded instructions to be a CIS.

reduction in the entropy of the reference sequences versus the sequences coded by a gene for a specific purpose defines the information gain.

CIS are often embedded hierarchically. The F_O region of ATP synthase is a component of the ATP synthase molecular machine, which is embedded in a mitochondrion, which is part of a cell, which is part of an organ, which itself is part of an integrated organism that contributes to an ecological CIS. Coded messages communicate intention between members of the system. In eukaryotes, many subsystems comprise an individual organism, whereas in prokaryotes there is more distribution of effort between collaborating species in an ecology with exchange of signals and genetic materials via passive uptake of DNA (Claverys et al., 2006), conjugal transfer, viral transduction, and other lateral gene transfer mechanisms (Stanton, 2007).

Indications Cells Could Be Computerlike

Modern computer architectures (Von Neumann architecture, n.d.) remind us of cells. DNA provides long-term storage, and the data are not randomly thrown together but sensibly structured, even as computers use file systems to organize related data. Genes in prokaryotes that need to be co-expressed are often located together and controlled by an operon (Osbourn and Field, 2009). In a recent study, for every eukaryote analyzed, gene order was not statistically random, but often those having similar and/or coordinated expression are clustered (Hurst et al., 2004; Michalak, 2008; Chu et al., 2011). Just as data on computer hard disks are stored in sectors, Alu-sequence containing nucleosomes define regions of the DNA (Salih et al., 2008; Trifonov, 2011).

DNA is a read/write/delete system. Data can be reorganized by transposons and content added via CRISPR (Clustered Regularly Interspaced Short Palindromic Repeats) (Zetsche et al., 2015; Ran et al., 2015), lateral gene transfer, and transfer of plasmids in prokaryotes. Genomes can also be contracted by deletions, such as the removal of transposable elements (van de Lagemaat et al., 2005). Portions of DNA are read many times and converted to mRNA copies—short-term memory—where logic processing is performed. Furthermore, mRNA codons specify amino acid sequences, so clearly a code exists.

We will focus here in Part 1 on formal software features like data types, data structures, codes, and algorithms, which are useful to solve problems using abstract methods, independent of the hardware implementation. The hardware aspects used by cells will be examined in Part 2.

Key Principles to Understand How Cells Work

Before showing that cells use formal software constructs, we need to devote some effort to eliminate a few misunderstandings and introduce some guiding insights: DNA does not provide an explicit prescriptive source program readable by humans; multiple codes are in use; each code requires a distinct alphabet and hardware decoder; software and hardware are far more integrated than in digital computers; and logic processing is distributed and hierarchical.

DNA Does Not Provide an Explicit Prescriptive Source Program

Many still erroneously believe DNA contains a prescriptive language containing a complete blueprint or “Book of Life:” that specifies in detail the development of an organism. As Woodward and Gills wrote recently, “This is the shock of shocks: that the DNA alone does not play the part of the director” (2012, p. 75). This contrasts with computer programs, whose logic can be understood from the source code. Consider as an example (1):

```
if (A=5 and B='red' and
not C='Deactivate')
then {'execute follow-
ing instructions'}     (1)
```

A line of readable coding such as (1) will not be found in DNA or elsewhere in a cell, but the variables can be identified, and logic operations are indeed being performed. Can we discern the Boolean logic and resulting processing being performed? Yes, empirically. Consider as an example of the variables A, B, and C three cis-regulatory elements (**CRE**s, specific nucleotide patterns on DNA). Each value is defined by which transcription factor (**TF**, a protein) is attached or "nothing is attached." The logic being performed can be deciphered by systematically varying the values (Davidson, 2006) and simulated with computer programs.

The logic is implicit but very real, and built into the system as whole, and for good reasons. Cells have far greater functionality than computers. They can replicate autonomously, generate their own energy, repair themselves, manufacture and recycle the substances needed, produce their processing hardware, and interact dynamically to provide emergent behaviors, even committing suicide (apoptosis) when necessary for the common good. An inheritable, error-free source code program to cover all these details and eventualities would not be feasible. Instead, cells replicate only the variables and their values, plus a functional copy of the necessary hardware each generation.

This strategy provides less opportunity for information corruption compared to specifying all the steps in precise detail in order to assemble thousands of cellular components, test the timing of location and progress of activity, and then mandate corrective action to take. We complete the explanation in Part 2 by showing how judicious organization—and inheritance—of the hardware components provide informative contributions and thereby reduce what the software needs to communicate.

Francis Crick was wrong when he claimed the genome was the (sole) source of phenotypic information (Crick, 1970). We can show this in many ways. A consequence of RNA editing, trans-splicing, and other post-transcriptional RNA modifications is that the modified sequences can undergo reverse-transcription and be introduced into the DNA germ line (Moller-Krull et al., 2008). Furthermore, changes in chromatin (which do not alter DNA sequences) can be inherited later over multiple generations (Jaenisch and Bird, 2003). In fact, somatically heritable chromatin structures are one way to establish differentiated cell lines (Gurdon et al., 2003). Further evidence that DNA does not directly prescribe final outcomes includes the existence of multiple life stages such as invertebrates having distinct larval and adult stages and other examples of metamorphosis. In the next paper, which accompanies this one, we describe the cell as an interacting set of controlling subsystems, each with its own coded variables, and less as a hierarchical or cascading design.

Much of what is necessary in the cell is not directly guided by DNA (Barbieri, 2003, p. 31). Globular proteins work only after they fold properly, which is affected by factors such as fluidity of the environment, how fast different sections are translated in a ribosome (Spencer et al., 2012), and the contribution of chaperones. Even after proteins form, additional guidance is provided, not by DNA, but by ligands, which are judiciously attached and removed. Gene regulatory networks develop automatically upon activating/deactivating CREs that are passively poised in anticipation. If one or more TFs activate a particular CRE, the resulting protein (a new TF) can activate or deactivate the same or different CRE(s), eventually leading automatically to mutually interacting circuits with no a priori guidance from explicit coded instructions.

RNAs can also behave as informative riboswitches. A small molecule binds to part of the RNA (the aptamer), which causes an allosteric change in another portion of the RNA called the expression platform, which can regulate gene expression (Serganov and Patel, 2007). There are many more examples of information processing that do not involve exclusive and direct guidance by DNA, such as aggregation of surface receptors in response to ligands (Wulfing et al., 2002; Bray and Duke, 2004; Murai and Pasquale, 2004) and cytoskeletal reorganization (Pollard and Borisy, 2003; Pelkmans, 2005).

There are cases, or course, where outcomes are partially specified directly by DNA, such as the N-end rule, whereby the half-life of proteins is determined to a large extent by the identity of its N-terminal residue. Sometimes DNA provides parameters less obviously such as in protein and vesicle targeting to distinct cellular locations (Bonifacino and Glick, 2004; Pool, 2005) and protein export from cells (Neel et al., 2005; Stuart and Ezekowitz, 2005). Here the signal sequences are extremely variable, both in length and amino acid composition, and the parameters are generated sometimes by remote parts of proteins brought together only after folding. This variability could be necessary for various processing details including additional post-targeting functions (Hegde and Bernstein, 2006; Emanuelsson, 2002; http://psort.hgc.jp/).

Evolutionists have generally argued that mutations are all that is needed to explain current cells. Distinguished Oxford professor Denis Noble, a forceful critic of Dawkins' reductionist views, pointed out that this is too simplistic: "Neo-Darwinism also privileges 'genes' in causation, whereas in multi-way networks of interactions there can be

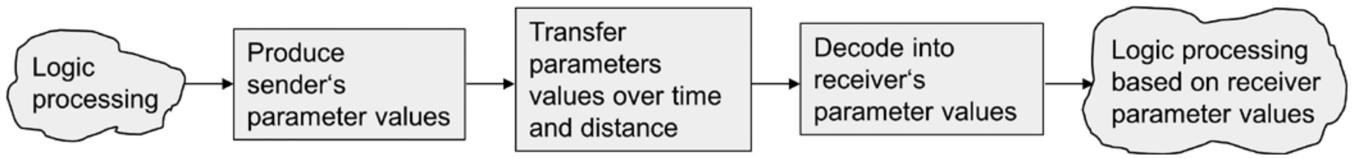

Logic processing
Produce sender's parameter values
Transfer parameters values over time and distance
Decode into receiver's parameter values
Logic processing based on receiver parameter values

Figure 2. Logic processing can occur by the sender before communicating coded data and after the receiver knows what should be done. Sometimes little or no reasoning is needed to generate the sender's data, such as photons landing on retinas, and thereafter complex logic must be executed by the receiver to extract benefit from the data and decide what is to be done.

no privileged cause" (Noble, 2015 p. 1).

Does DNA determine outcomes by already possessing the necessary instructions, or does it respond to signals from the cell (e.g., to replace proteins decided by the cell are needed)? We agree with Noble, who also wrote, "The causality is circular, acting both ways: passive causality by DNA sequences acting as otherwise inert templates, and active causality by the functional networks of interactions that determine how the genome is activated" (2015, p. 9) and that "IF-THEN-ELSE" type instructions are found in cells (p. 10).

An interesting consideration is where most of the decision making occurs in computers and cells (Figure 2). This issue arises in all sender-receiver forms of communication. In some cases, a message could provide very detailed instructions, and in other cases the message is (explicitly) minimally informative. When only variable values are communicated, sometimes the sender performs considerable logical preprocessing and then only provides what is relevant (which the receiver can easily process). In other cases, raw data are made available, and the receiver is responsible to make sense out of it.

In the first example, we will consider, the *sender* has performed most the important logic processing before sending the following coded data (2):

```
(Co='IBM'; Nr_stocks_
to_buy=510; When_to_
buy='16 o'clock CET')
```
(2)

The receiver now knows what to do (which stocks to buy, how many, and when). Considerable decision making occurs in cells in the sender environment before the concentration and location of TFs are specified, and the results are communicated and processed as variable values by the relevant CREs variables at the receiving side.

In the next example, the *receiver* must perform much deductive processing, since variable values are communicated whose significance need to be interpreted and evaluated (3):

```
(Co='IBM'; Stock_
change_in_price=0.1;
Weather='cloudy';
Winner='Manchester
United')
```
(3)

The receiver must now determine what is relevant and how it correlates with the decisions to be made. Human minds typically process raw data considerably before making a decision.

What Is a Code?

What is a code, and how does it relates to logic processing using variables? A code defines rules that translate physical or mental details—such as sounds, images, pressure, size, quantity, intention, or even a different code—between two independent systems using an agreed-upon abstract convention of symbols. Speaking and writing are examples, bridging gaps in location and time. A simple causal outcome based on only a mechanical effect does not use a code, so an axe blow does not split a log in two thanks to a code that communicates intention. Whether to swing an axe could be communicated, however, using an arbitrary symbol convention such as {thumbs up /thumbs down}, {0 / 1} or {oui / non}.

The sender and receiver can share the same symbol set (alphabet), like the International Flag Code for merchant ships and the use of 'Co' in (2) and (3) above. An example in cells is when a specific TF (sender value) interacts directly with a CRE (generating a receiver value). Another example is when a DNA template is used to generate DNA copies. The next nucleotide value to add to the growing chain is communicated directly.

Alternatively, the sender and receiver could use different alphabets and variables as long as there is an unambiguous way to map the symbol sets. In (2) above, the receiver could assign the value for "Co" to its own variable "Company" and also convert the time 16 o'clock according to its own time zone. This kind of linkage may require **adaptor molecules**

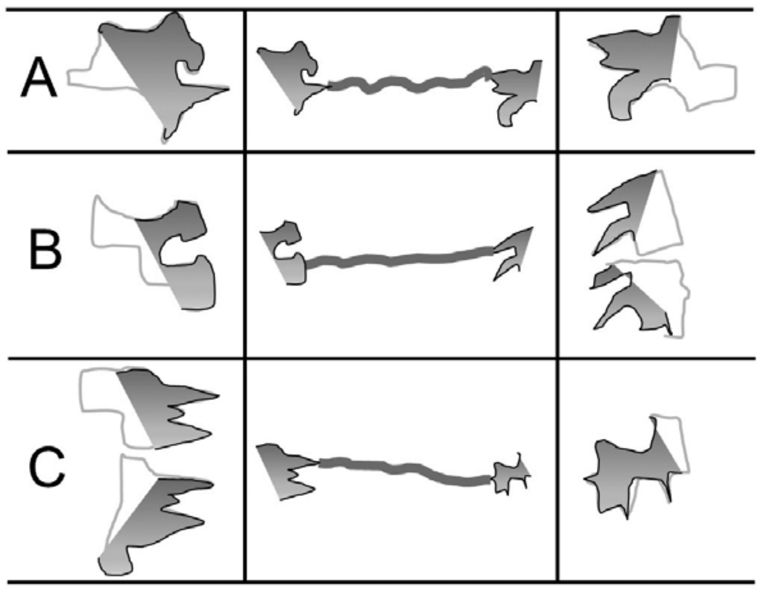


Figure 3. Communication between a sender and receiver system corresponds to transferring values to receiver variables. The alphabet of the sender (dark symbol in leftmost column) can differ from that of the receiver (dark symbol in rightmost column). In cells one or more adaptor molecule (middle column) may be needed to translate values between sender and receiver variables. The correct adaptor is identified through physical linkage with the sender variable's value.

A. 1:1 mapping between sender and receiver variable.
B. 1:n mapping.
C. n:1 mapping.

(Figure 3) or messenger molecules in cells.

If an informative ligand attaches to a TF, which then links to a CRE, this TF is now playing the role of an adaptor molecule. Another example of adaptor molecules are tRNAs in the genetic code, where one end identifies a specific mRNA codon (the value), and the other end translates to a corresponding receiver value (which activated amino acid to add to the growing protein). Linking the two systems through a chained network of signals permits additional factors to be taken into account that could refine the details during transfer.

Additional variables can be used within the sender and the receiver side to perform necessary reasoning. These can be independent codes, but at their interface there must always be pre-agreed conventions with respect to the meaning of the variables and how values are communicated. A receiver could then process the values assigned to its internal variables and then become a new sender, transmitting values to a new receiver. A chain of sender/receivers can result, and examples in cells include signal cascades.

Multiple Codes Are Used in Cells

Gordon Tomkins may have been the first scientist to propose that the genetic code is not the only code used in biology (Tomkins, 1975). Cell needs are communicated by different codes found on DNA, RNA, proteins, filaments, sugars, cell membranes, and other cellular components. Occasionally the literature seems to incorrectly claim a code is involved, such as the so-called **protein folding code** (Dill et al., 2008), in which multiple local activities occur in a precise order as part of the folding process. The difficulty in this case is identifying abstract variables upon which Boolean logic is performed. In this example, it seems that only physical chemical forces are occurring in a continuous and time-ordered set of steps. No variables are waiting to be assigned values nor anticipate activation.

Each code has its' own language and symbols. The **genetic code** to specify protein sequences is independent of the **DNA-binding protein code** to regulate gene expression (Hughes, 2008; Jolma et al., 2015) (Figure 4), even though both use DNA, and DNA codes sometimes share overlapping DNA nucleotides.

Entire collections of CREs can be organized into cis-regulatory modules (CRMs), leading to DNA code variants, since each CRM uses a separate set of rules. Figure 5 shows a representative example, where three exon are regulated by five such CRMs (Davidson, 2006, p. 49). Depending on time (e.g., development stage), input signals, and cell lineage, different modules can be used to interact with the key "proximal module" nearest to the transcription apparatus. This is a clear example of Boolean logic being applied.

In addition, by using different reading frames, the same code sometimes provides different messages. This was examined in mathematical detail for the genetic code at a recent conference on biological information (Montañez et al., 2013, pp. 139–167). In a remarkably candid paper, we read that "although dual coding is nearly impossible by chance, a number of human transcripts contain overlapping coding regions" (Chung et

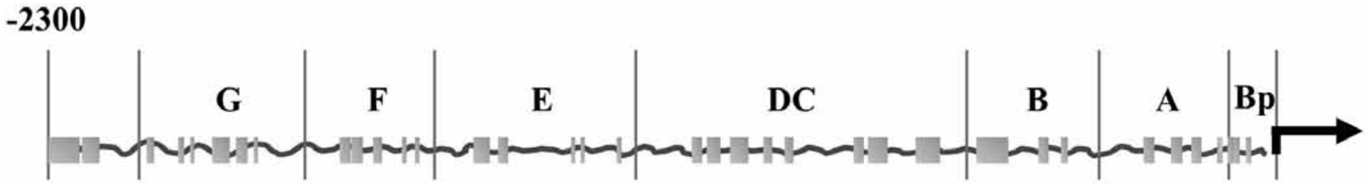


Figure 4. Representative example of cis-regulatory logic, showing the 2300 base-pair region preceding the coding region of gene endo16 of sea urchin. One or more proteins can bind to each of the cis-regulatory elements (gray boxes). The letters identify regions used for different purposes, such as regulation of key tissues during different phases of development (Davidson, 2006, p. 49–51).

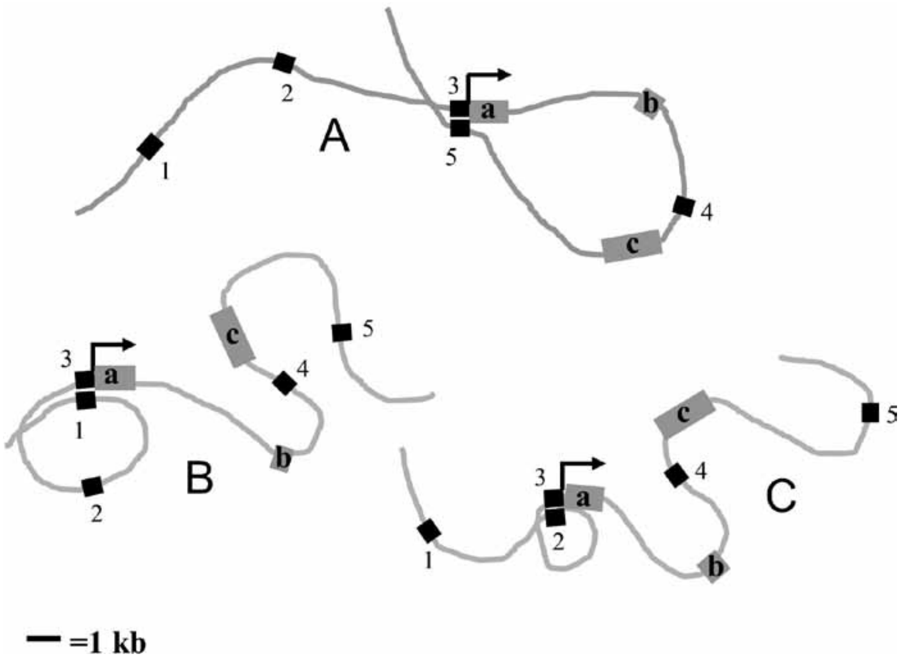


Figure 5. Multiple cis-regulator modules (CRMs) per gene, each composed of several CREs, permit independent regulation according to time, input signals and cell lineage. This typical example shows three exons (gray-checked boxes) regulated by five CRMs (black boxes). The CRMs are about 400 bp long, and the gene plus regulatory regions are spread out over about 30 kb of DNA. Alternative looping brings the relevant regions together (Davidson, 2006, p. 49). A: The "proximal module" 3 interacts with CRM 5; in B it interacts with CRM 1, and in C it interacts with CRM 2.

al., 2007). These multiple codes prompted Trifonov to point out, "The times of surrender to 'junk' and 'selfish DNA' are over, and 'non-coding' becomes a misnomer" (Trifonov, 2011, p. 2).

We will not attempt an exhaustive listing of all cellular codes at this time, and the DECODE program continues to bring new ones to light, but we will mention a few to demonstrate that cellular codes define variables and their values but not procedural code as humanly readable instructions.

There is a **tRNA charging code** without which the genetic code cannot be implemented (Hou and Schimmel, 1988; Trifonov, 2011).

The **histone code** (Young, 2001; Jenuwein and Allis, 2001; Strahl and Allis, 2000; Cosgrove and Wolberger, 2005) involves post-translational modifications such as ubiquitination, phosphorylation, mono-, di-, tri-methylation, acetylation, sumoylation, and biotinylation of various residues on the four histones proteins (H2A, H2B, H3, and H4) that form the nucleosome. These tags regulate gene expression and other processes. Specific histone modifications can identify the need for DNA mismatch repair, for example H3K36me3 (histone H3, lycine number 36 receives three methyl groups) (Schmidt and Jackson, 2013) and H3K56 acetylation (Kadyrova et al., 2013). Hypoacetylation of H3K56 by enzymes HDACs 1 and 2 facilitate recruitment of nonhomologous end-joining (NHEJ) proteins (Miller et al., 2010). One should not overlook that each cell type in eukaryotes uses its own histone code (Carey, 2012, p. 188).

DNA methylation at the correct location identifies which sections of

DNA should be transcriptionally active euchromatin or inactive heterochromatin (Bird, 2002).

The **tubulin code** involves various ligands that are added and removed to microtubules to affect several cellular processes (Verhey and Gaertig, 2007; Janke, 2014).

The **splicing code** of eukaryote pre-mRNAs permits different exons to be combined to produce alternative proteins (Tejedor and Valcárcel, 2010).

The **nucleosome positioning codes**, also called "Chambon rules" (Barash et al., 2010), are understood well enough to algorithmically automate their location to within one base for biological DNA sequences (Segal et al., 2006; Trifonov, 1980; Trifonov, 1981; Gabdank et al., 2010). During development eukaryote genes are activated in a timed based manner using these codes for each primary transcript (Segal et al., 2006) to establish a regulatory circuitry that controls which genes are activated or silenced (Yuh et al. 1998).

Interaction between genes has also revealed the **Hox Code**. Just a few Hox or homeotic genes control development of the body plan along the anterior-posterior axis. They code for transcription factors, which can either activate or repress large gene networks. The same transcription factor can repress one gene and activate a different one, and TFs are involved at many levels within developmental processes (Wellik, 2007). A typical regulatory region in eukaryote DNA is about 500 nucleotides long, on which four or five transcription factors can bind. On average eukaryote genes seem to have about three such regulatory regions (Bray, 2009, p. 191).

The **N-end code** regulates the half-life of a protein using the identity of its N-terminal residue, which is determined from the moment they are produced (Varshavsky, 2011; Gibbs et al., 2014).

In the **sugar code**, oligomers of carbohydrates serve as ligands for the transfer of information, acting with lectin protein receptors (Gabius et al., 2011; Murphy et al. 2013). The large number of hydroxyl groups available offers enormous storage capacity, vastly more than the genetic code could (André et al., 2015).

The **adhesive code** (Readies and Takeichi, 1996; Shapiro and Colman, 1999) uses differences in adhesiveness between neural cells in the primordial neuroepithelium to first establish segmentation and then the emergence of specialized structures such as brain nuclei, cortical layers, fiber tracts, and neural circuits using cadherins.

A **niche code** has been proposed (Forsberg and Smith-Berdan, 2009). Hematopoietic stem cells (HSCs) must generate daughter HSCs and a variety of mature cells in response to stress in a regulated manner. HSCs are found in specialized niches in bone marrow, and there is a regulated adhesive interaction between niche cells and HSC components such as integrin, another example of adaptor molecules.

Signal Transduction Codes are used when extracellular signals ("first messengers" such as hormones, neurotransmitters, and paracrine/autocrine agents) attach to a specific receptor on the cell membrane, activating a smaller number of second messengers such as calcium, cAMP, nitric oxice, and phosphorylation cascades (Figure 6). One signaling molecule can cause many responses such as the cell's metabolism or gene expression, an example of 1:n variable mapping mentioned in Figure 3).

There is a vast research literature on this topic, and resources on signal transduction pathways are available on-line in databases such as "NetPath" for humans (http://www.netpath.org/). The latest research is correcting the view that simple linear cascades are used. Instead, large networks consisting of hundreds or thousands of proteins are involved (Walhout et al., 2013, p. 93). Note the rich potential to interact with other networks and codes to dynamically integrate multiple cell inputs and needs.

The actin cytoskeleton uses adapter molecules to identify materials that should interact there, which implies a **cytoskeleton code** (Barbieri, 2003; Barbieri, 2008, chapter 8).

The complex firing of neurons in the brain uses some kind of **neural code** or codes, since meaning is gleaned that permits the internal and external world to be understood (Nicolelis and Ribeiro, 2006; Cessac et al., 2010; Jessell, 2000; Marquardt and Pfaff, 2001; Flames et al., 2007). In spite of intense interest, it is far from being understood.

A **phosphorylation code** in Hedgehog signal transduction has also been identified (Chen and Jiang, 2013; Ficz, 2015; Schübeler, 2015).

The **miRNA code** can up or down regulate individual mRNA levels according to eukaryote cell type (Carey, 2012, pp. 191–194).

A **CpG epigenetic code** in eukaryotes governs millions of methylations on DNA. When near the gene start site, transcription is blocked but in the gene itself enhances expression (Jones, 2012). In this read/write/delete system, DNA-methyltransferases (DNMT) add methyl groups, and there are many mechanisms to remove them in a tissue-specific manner. Methylation is most dramatic in the brain (Keverne et al., 2015). Most of the methyl groups are removed in the fertilized egg (zygote) (Lee et al., 2014), otherwise the next generation would begin with a specialized and not pluripotent cell.

The **ventral neural tube** is an example of special codes used in cells that interpret a gradient concentration. Distinct classes of neurons are produced in the ventral neural tube according to local concentration of Sonic Hedgehog (Shh) (Briscoe et al., 2000).

Many secreted and membrane proteins contain N-terminal **signal sequences** that communicate their target locations (Hegde and Bernstein, 2006).

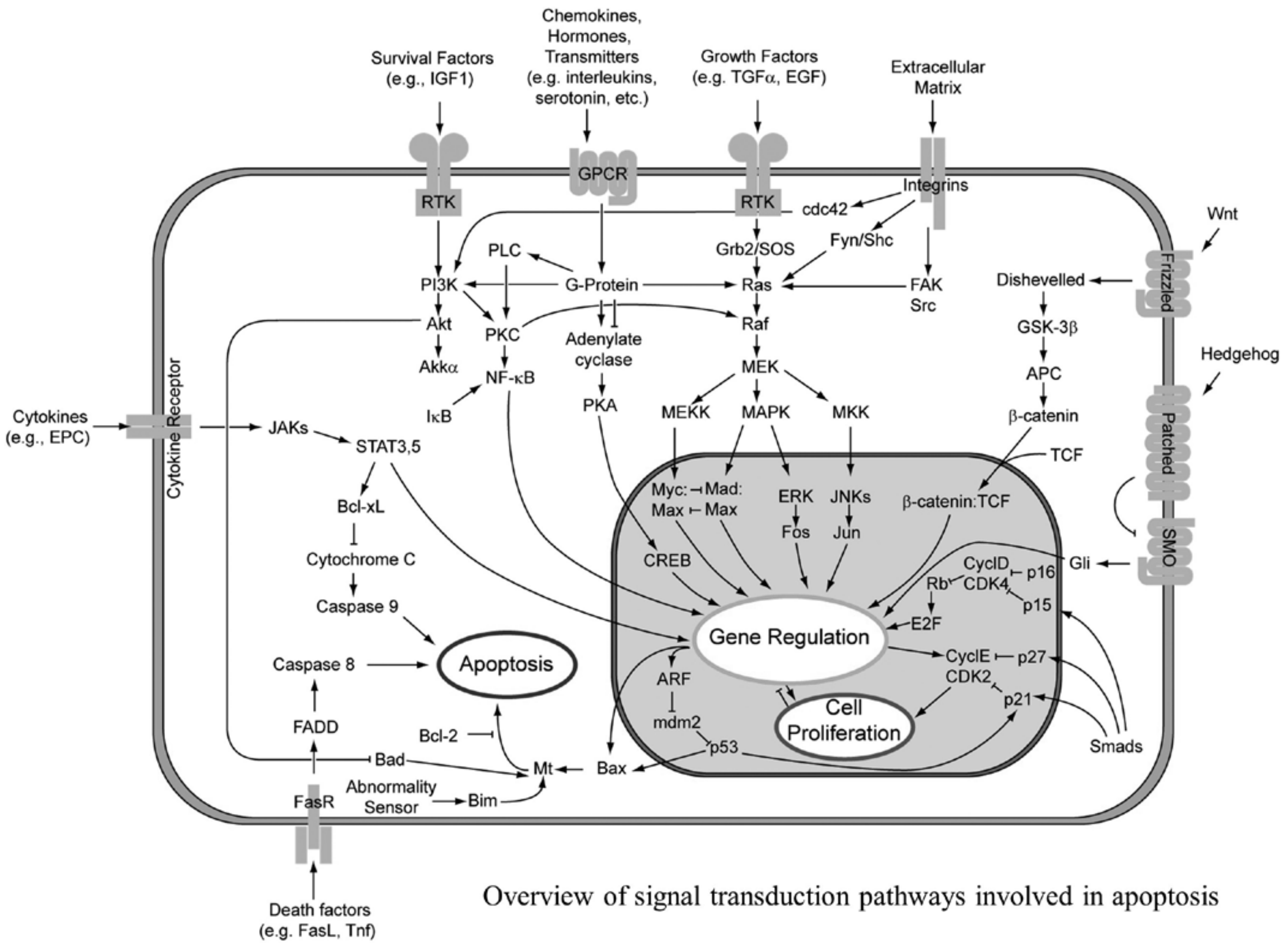


Figure 6. Example of a signal cascade pathway, here involved in programmed cell death (apoptosis). (Source of diagram: Wikimedia Commons, the free media repository, https://commons.wikimedia.org/wiki/File:Signal_transduction_v1.png). See also Klipp et al., 2009, pp. 135–142.

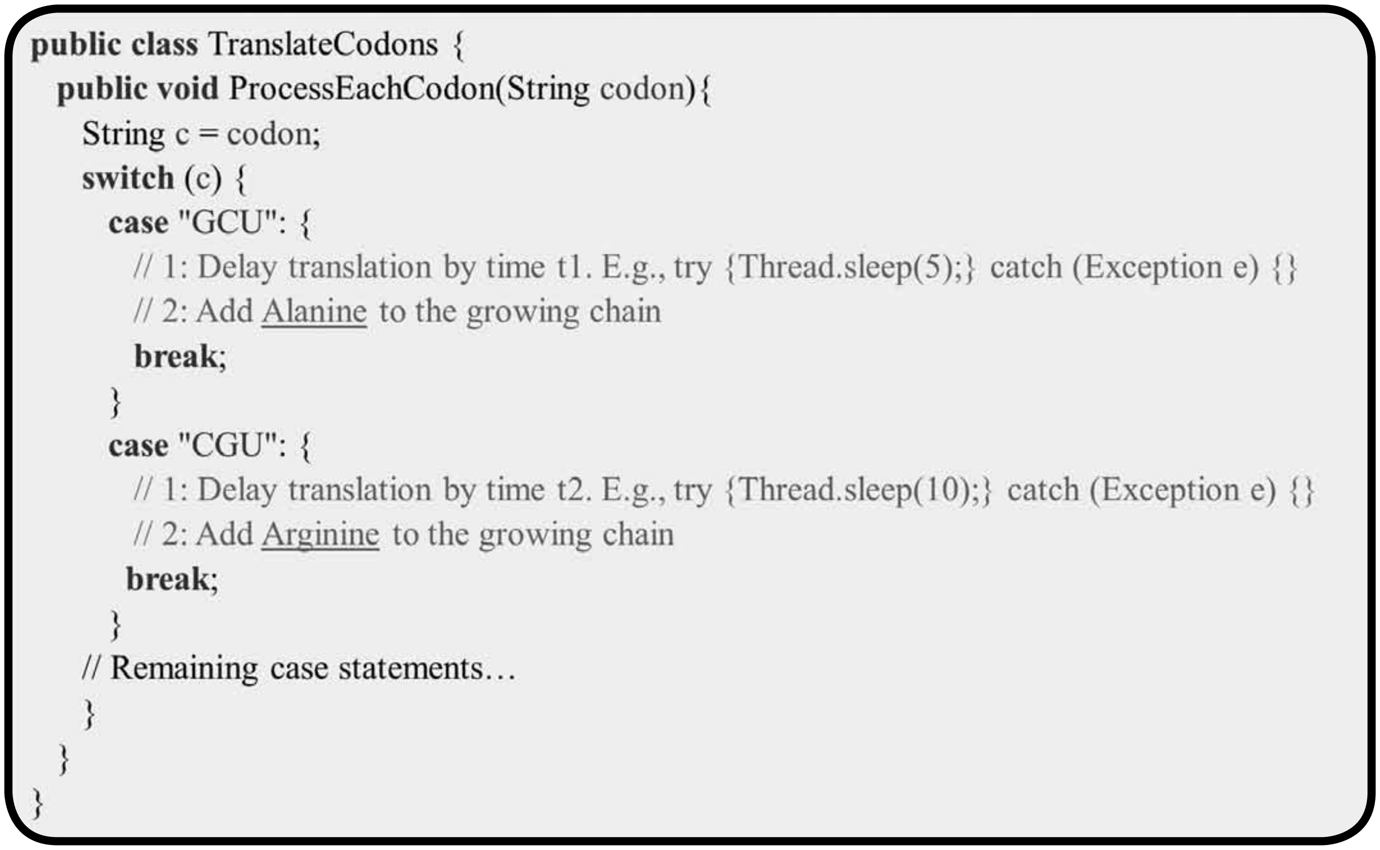
Codes in Cells Can Overlap

Cellular codes often overlap and therefore require degeneracy to not overly restrict each other. Since codes can be implemented using biochemicals which themselves rely on the genetic code, complex design tradeoffs are necessary. When planned correctly, the best implementation must be as robust as possible, taking into account the severity of possible errors for all the affected codes (mutations, mistranslation, etc.).

Degeneracy with respect to one code could be critically important for a different one. As an example, different codons could represent the same amino acid in the genetic code, but each codon can specify how rapidly that position is translated. Figure 7 describes this using a section of Java programming.

In probably all cases, assuming complete degeneracy for a code would be a mistake. Variants of a class of CRE could all be recognized by the same TF, but the CRE sequence differences specify how long and often to remain attached, in which tissue type, the timing of activity during a cell cycle, and for what stages of development.

The use of multiple and overlapping codes saves material and energy but is too constraining and requires too much foresight to find applicability in general purpose programming by humans.



```
public class TranslateCodons {
  public void ProcessEachCodon(String codon){
    String c = codon;
    switch (c) {
      case "GCU": {
        // 1: Delay translation by time t1. E.g., try {Thread.sleep(5);} catch (Exception e) {}
        // 2: Add Alanine to the growing chain
        break;
      }
      case "CGU": {
        // 1: Delay translation by time t2. E.g., try {Thread.sleep(10);} catch (Exception e) {}
        // 2: Add Arginine to the growing chain
       break;
      }
    // Remaining case statements…
    }
  }
}
```

Figure 7. Java example of codons being used for two unrelated purposes: to determine amino acid sequence and translation rate at that position of the mRNA.

Each Code Uses Its Own Processor

It is important to understand the distinction between variables and the values they can assume. Cellular variables possess recognizable steric and electronic features and wait for activation by a sender (which provides the values). For example, transcription in bacteria through RNA polymerase involves variables, like the “sigma factor recognizing promoter” (the -35 and -10 elements located before the beginning of the sequence to be transcribed). As possible values these locations could be unbound or bound to one of several possible “sigma factors.” The sigma factor can also interact with a distinct set of promoters (Ishihama, 2000).

For each coding system there are special processors designed to interpret the relevant values. When TFs bind to cis-elements to regulate translation, an appropriate three-dimensional processor involving many proteins must be organized which can include direct or indirect adaptors (Zhou et al., 2015). The hardware aspect of cellular design is discussed in Part 2.

Software and Hardware Tightly Integrated

Unlike a Turing or von Neumann Machine (Von Neumann architecture, n.d.), cells must repair themselves, generate their own energy, adapt to new challenges, and reproduce autonomously with all necessary components over many generations. The solution is a complete synergistic interaction between the software and hardware. The physical DNA, RNA, and protein-based components that produce the hardwired biochemical processes are themselves constructed and replaced by relying on data provided through preexisting DNA, RNA and proteins.

It is often easy to identify the physical components of cells but overlook informational aspects. Each 260 million photoreceptors on a human retina could be identified, but the semantic content implied by the photons landing on them is then funneled on to only 2 million connected ganglion cells before sending to the correct processing regions of the central nervous system (Gazzaniga et al., 2009). Here information is being interpreted, compressed, and transferred.

As a second example, microtubules do much more than only maintain a cell's shape. Per microtubule a hundred thousand or more globular protein units grow in many directions and degrade constantly until coming into contact with a specialized region of a chromosome centromere (Sullivan et al., 2001), or membrane, after a signal arrives there, at which point a firm attachment prevents degradation (Kirschner and Gerhart, 2005, pp. 148–152). These attachment regions are sensors (variables) that assume a value (i.e., when activated by the tip of the microtubule), that recruits proteins to produce a decoding molecular machine.

Logic Processing Is Distributed and Hierarchical

Different prokaryote species form ecological systems with necessary genes distributed among the members (Sonea and Mathieu, 2001), which is why a particular function requiring several genes can be assembled in one member through horizontal DNA transfer. Plasmids in prokaryotes are another example of **distributed** information processing. In eukaryotes, information processing is also distributed, such as when bacteria digest food separately from the host organisms' germ line. Different cell lineages also distribute the effort, where each cell type has characteristic ensembles of activated and deactivated genes. Proteins, polysaccharides, lipids, and other substances are used to interact with receptors on cell surfaces and provide communication signals to convey metabolic and developmental status back and forth (Aricescu and Jones, 2007; Takada et al., 2007; Yamada and Nelson, 2007; Widelitz, 2005). Intercellular communication also occurs by molecular diffusion through air or water using gases, amino acids, oligopeptides and vitamins as signals (Bogdan, 2001; Chen et al., 2005; Fuqua et al., 2001; Chambon, 1995; Lazazzera, 2001).

Hierarchical information processing also occurs. As examples, low-level logic processing occurs when individual DNA nucleotides define individual RNA nucleotides, and when codons specify amino acids. Once a protein has formed, additional processing occurs to transfer it to the correct cell location, later to integrate into molecular machines, enzymatic networks, and metabolic networks. Thereafter ever more complex features can develop, such as entire eukaryote organelles which themselves become part of a properly regulated cell, on up to organs, whose operations must also be carefully regulated to permit a viable organism that interacts within an ecology.

In addition to such hierarchical integration, we will see in the accompanying paper that many control systems in cells—each with their own codes—interact mutually within what often seems to be the same hierarchical level.

Generic Insights from Computer Systems Architecture

The explosive development of computer technologies is the result of collaboration between millions of scientists, engineers, and mathematicians worldwide. Fundamental to this success are interoperability conventions and standards (such as the Open Systems Interconnection model). This permits specialists in various hardware and software areas to focus on and develop technologies from which integrated systems result. Using these design insights, we will interpret cellular behavior by examining software and hardware aspects individually and consider different levels in the system at which guidance is provided.

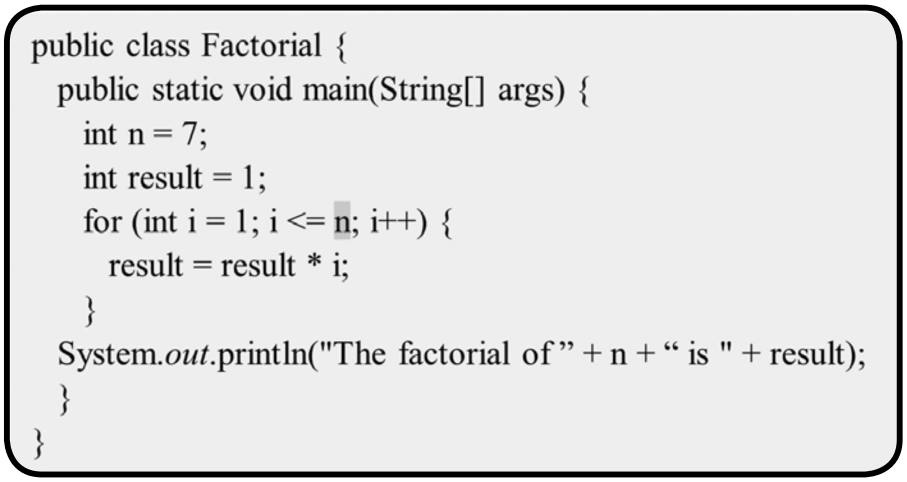
Another insight humans have gained is the design of subsystems that can be assembled. We discussed lateral and hierarchical logic processing above (Thanbichler and Shapiro, 2008; Schneider and Grosschedl, 2007). An external printer can be built separately and then linked to the rest of the system. To work properly the hardware devices often also require their own dedicated software (e.g., "drivers" must be installed).

Software Elements Used to Implement Processing Logic

Before examining software constructs used by computers and cells, let us consider a simple program to calculate the factorial of a number (Figure 8).

Several general principles can be discerned.

1. The programmer did not need to consider how the solution would be implemented on hardware nor the operating system details. Only the logic needs to be accurately expressed symbolically.
2. There is a language with a precise grammar that contains several generic constructs—for example, iteration (with a defined starting and finishing value) and a *Boolean test* (if *i* has a value of *n* or less, then add 1 and continue, otherwise terminate the iteration).
3. The same processing logic could be applied with different values and meanings for the variable *n*.
4. The algorithm could be copied into other programs and modified.
5. The variables belong to a specific data type and have properties consistent with them. In the example, *i* and *n* must have an integer value: one cannot assign a value of "Smith" nor "True" to them.
6. The variables can represent real objects, like dollar bills, but the choice of the symbols and what they do are physically independent of what they specify.
7. The algorithm continues to make sense if each variable is replaced by another unique symbol. Even a three-dimensional abstract symbol could be used and the values assigned could also be represented by no code currently in use by computers. However, changes in hardware would then become necessary.



```
public class Factorial {
  public static void main(String[] args) {
    int n = 7;
    int result = 1;
    for (int i = 1; i <= n; i++) {
      result = result * i;
    }
  System.out.println("The factorial of " + n + " is " + result);
  }
}
```

Figure 8. Programming using Java to calculate the factorial of a number to illustrate the use of common software constructs to solve problems independent of the hardware implementation.

8. To have any value, the outcome from the algorithm needs to be retained or have some kind of effect.

All these and other principles can also be identified in cellular information processing. In the example in Figure 8, we see how limitless cases could be solved by merely replacing the numbers *i* and *n* as needed. This works only if programming constructs such as iteration, assignment of values to variables, and so on, exist. Otherwise a unique mechanical arrangement would be needed to solve each example. It is this use of general-purpose symbolic logic, which can be mapped to mental or physical objects, that is so special about computers and cells.

After this long, but necessary, preparation, we are finally ready to examine three important topics in the art of designing software: generic software data structures; generic programming elements; and file formatting. These are fundamental for computers.

I. Generic Software Data Structures.

Let us examine how data is usually structured in modern computers and cells to facilitate use in general-purpose programming constructs discussed in section II.

Symbols in an alphabet

Codes rely on an alphabet of elementary symbols. Modern digital computers use an alphabet of two symbols {0, 1} called bits. Cells use dozens of alphabets for their many codes. DNA is composed of four nucleotides abbreviated {A, C, G, T}, RNA also uses four nucleotides {A, C, G, U}, other codes rely on small ions such as cAMP (Ashcroft, 1997; Krysko et al., 2005) and calcium (Wagner et al., 2015), or on small parts of larger molecules.

One or several symbols taken jointly define an item, field, constant, variable, or value. In the past, telegraph messages used 5-letter commercial coded values such as BYOXO ("Are you trying to weasel out of our deal?") and LIOUY ("Why do you not answer my question?"). Other conventions also exist, such as LOL ("Laughing Out Loud") and CU ("See You"). In the extended ascii ISO 8859–1 code, '00001001' represents a *Line Feed*, '01000001' represents the letter A, and '00111000' represents the decimal digit 8. The codeword length of values can be fixed as in the asci extended and the genetic code or have different lengths as in compressed codes to store and transmit electronic data (Togneri and deSilva, 2003). There are design trade-offs to consider when deciding whether to use a fixed or variable length (Truman, 2012).

The symbols used by computer programs must be exact to be processed. C*onfirmation* and C*onformation* are almost identical, but not the same.

Different codes can be linked using different alphabets. A sender code could be restricted to a symbol from, e.g., {green, yellow, red}, which the receiver could translate to its system, e.g., limited to {1, 2, 3}.

When large molecules are used to convey coded meaning in cells, typically a small portion is informative, and the rest plays an adaptor molecule role or is used for the implementation details. Consider proteins. Portions of different residues are integrated to define a joint "symbol" having unique steric and electronic properties. The resulting symbols must be decoded using three-dimensional processors. In the fluid environment of cells under varying temperatures, the decoders must be more flexible than in computers. One consequence is that a portion of different amino acids could be combined to produce functionally the same symbol meaning in three dimensions.

Data types

Modern computer languages enforce data typing, which defines the kinds of values that can be assigned to variables to prevent errors. Common types include integer, floating-point number, character, alphanumeric string, and Boolean. Each kind of variable for biological codes is restricted to a range of values. The genetic code uses DNA and mRNA codons, whereas the enzyme complexes used by the histone code do not process codons (http://www.cellsignal.com/contents/resources-reference-tables/

histone-modification-table/science-tables-histone).

As another example, many mRNAs can interact with only some miRNAs (which specify what is to occur to the mRNA; Verdel et al., 2009; Sugiyama et al., 2005). This corresponds to 1:n, n:1, or n:m variable binding in Figure 3. In addition, only certain noncoding RNA data types (specific siRNAs, piRNAs, Alu RNAs etc.) are recognized by mRNA binding proteins.

Data type subsets

A subset of a data type can also be established for a specific program or module to further narrow acceptable values in some programming context—for example, only certain acceptable city codes for telephone numbers in a city, or a list of alphanumeric identifiers for a product line. We find this principle also in cells. The codons to represent Alanine must come from the subset {GCU, GCC, GCA, GCG} and Arginine from {CGU, CGC, CGA, CGG, AGA, AGG}.

Operation are defined for each data type

Specific computing methods or operations are permitted for each data type (and also for complex structures like matrices, arrays, etc.). One can negate a Boolean variable to convert True into False, but negating a data type "character" makes no sense. String variables can be concatenated, for example phrase = "white" + " " + "house" to form "white house," but this won't work for variables such as integers.

This principle is also found in cells. Each code used with DNA, RNA, proteins, sugars, or membranes is limited to its variable type(s) and their allowed operations. Consider the processing operations that can be performed with mRNA's data type "codon." The values can be read at the A (acceptor) or E (exit) portion of ribosomes (the receiver variables), they can be "concatenated" on each side to form polymers, and they can base pairs in unique ways (A-T and C-G). These kinds of operations cannot be assumed for other data types, such as hormones, transcription factors, or neurotransmitters. Ribonucleases and restriction enzymes can cut DNA strands using a subset of acceptable patterns (the receiver variable), but these locations are not processed on a codon basis as the genetic code does.

Group item

Elementary fields or items in computer languages can store values long-term using compound symbols. In many programming languages, several elementary items can also be combined and processed jointly for read and write purposes. As an example, a group item "*address*" could be composed of elementary items "*house-number*," "*street*," "*city*," and "*country-code*." Additional hierarchical clustering is also used in computer languages (such as C, Pascal, and Cobol), meaning group items can be further combined into records for example. This principle is also found in data transfer conventions like XML.

In cells, we recognize this principle whenever elements containing substructures are processed as a complete entity. One example is telomeres at the end of chromosomes, composed of groups of repetitive nucleotide patterns (e.g., TTAGGG in vertebrates), which are replenished by the enzyme telomerase reverse transcriptase. The six individual nucleotides are processed as an ensemble. In ***S. cerevisiae***, each $C_{1-3}A/TG_{1-3}$ repeat, taken jointly, constitute a potential binding site for Rap1 proteins, which recruit additional proteins (Williams et al, 2010).

In mammals, shelterin protein complexes regulate telomerase activity. Two of the six subunits (TRF1 and TRF1) bind uniquely to individual double-stranded TTAGGG (de Lange, 2010). So once again we recognize the concept of a grouping of elementary components. At a higher level, multiple copies of the individual patterns are treated as a new grouped entity and added all together to a chromosome by TERT (TElomerase Reverse Transcriptases) using a piece of template RNA known as TERC (Jády et al, 2006).

Group items consisting of smaller group items are not limited to repetitive patterns. Multiple codons are placed together within exons, which themselves are integrated into a primary RNA transcript. Processing as a whole occurs, such as in retrotranscription and rearrangements with the help of transposable elements.

The concept of group processing reminds us of how several residues jointly lead to discrete motifs in folded proteins and how a larger numbers of residues work together to form secondary structures such as alpha helices and beta sheets. Different nucleotide combinations also produce special RNA motifs.

Microbial genomes are also known to have an operon-like organization at various scalar levels (Audit and Ouzonis, 2003).

Concatenated index

In relational databases such as Oracle, a unique combination of one or more index values can be used to identify data records. Similarly, multiple nucleotides define promotors to identify the location of genes.

Array

Arrays and linked lists contain a series of values. In arrays, values of some datatype are stored in numerically indexed positions. The position within the array is informative and can be used directly in programming logic. If a certain value is always located at a specific index position (or a limited range of positions established in advance), it can be accessed directly by processing logic. An example using Fortran (a language well-suited to matrix calculations) is shown below (4). Assume that the results of a student's different exams are stored in known index

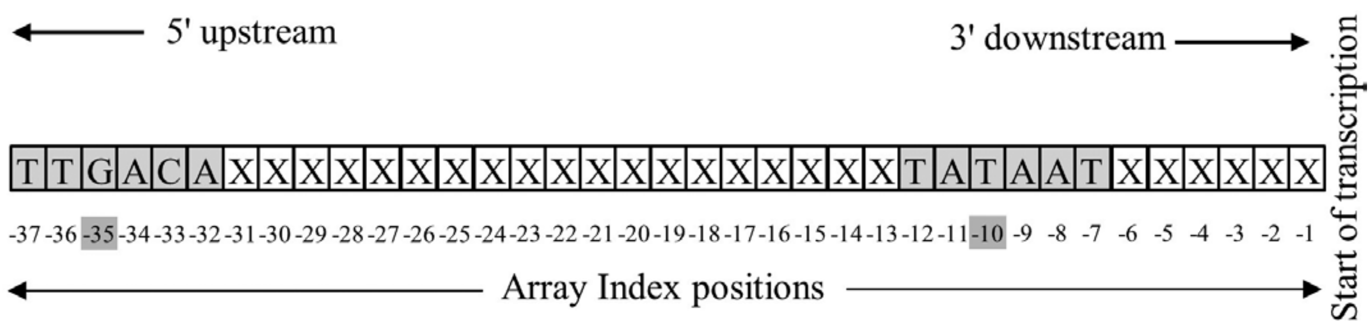


Figure 9. Nucleotide patterns at specific locations in bacteria define consensus promoter elements. The Pribnow box is centered at the -10, and a second component is often found at the -35 nucleotide position upstream from the start of transcription. Other regulatory elements are sometimes centered at the -41 or -61 position. If each nucleotide in the regulatory region is stored in an array, the index position can be used to program logical tests.

positions of array Examresults, and index position 3 contains the points obtained for the math test. The programing logic might look like this:

```
IF (Examresults(3) .GE.
70 .AND. Examresults(3)
.LT. 85) THEN Mathgrade
= 'B'                    (4)
```

Highly relevant to our discussion about cells, the value of interest could in principle be stored in different array positions if the acceptable alternatives are established in advance. Suppose there were two examiners and the result if determined by the first one is stored in Examresults(20) and if by the second examiner in Examresults(21). Now the program must determine the test results for the math exam by looking up the contents of array positions 20 and 21 and select the one having the exam result.

Prokaryote promotors illustrate array data storage and processing. For the Pribnow Box, a six-nucleotide consensus TATAAT is used by E. coli, centered at the -10 position, and often a second pattern TTGACA centered at -31 (Figure 9). For some bacteria or genes, the array positions to check could be slightly shifted, but legitimate indexed positions to be tested are known in advance. We will not elaborate here on the reasons for using alternative array positions, but it could be to regulate transcription rates or the results of genomic rearrangements.

There are many more examples of array processing in cells. In a typical ca. 22-nucleotide miRNA, usually only 6–8 adjacent or almost adjacent nucleotides (the seed region) at the 5' end are relevant, which is also true of the corresponding receiver variable on an mRNA. Logical tests on candidate miRNAs and their binding sites can therefore be performed using array index values. As another example, the coding parts of DNA and mRNA specify amino acid sequences, and the nucleotides need to be processed as triplets with no frameshifts. This permits translation to read the codons located in sequential index positions along mRNAs. In other words, each array index position does not contain a nucleotide, but a codon. Once the mature mRNA is ready for translation the length remains fixed, another characteristic of arrays.

Additional examples of processing array data include the symbols used by mobile elements to recognize insertion motifs; the portions of folded TFs that recognize cis-regulatory combinations; and the portions of enzymes that recognize restriction sites.

We see why many proteins must fold reliably into the same three-dimensional structure. This brings the relevant elementary symbols together so each can be assigned to a three-dimension index, "protein_position[i,j,k]." The relevant array positions refer to location in three-dimensional space and not the primary protein sequence. The resulting symbols need to be defined well enough to permit variables and their values to recognize each other, synergistically molding themselves together and avoiding false positives.

Whenever for a DNA or RNA-based code the distance between key nucleotide patterns are exactly or almost exactly known (including epigenetically modified nucleotides), then an indexed array seems to be a better description than a linked list. Knowing index values allows other array positions to be skipped and ignored. This is physically implemented in cells by constraining the decoders (e.g., portions of proteins) to specific

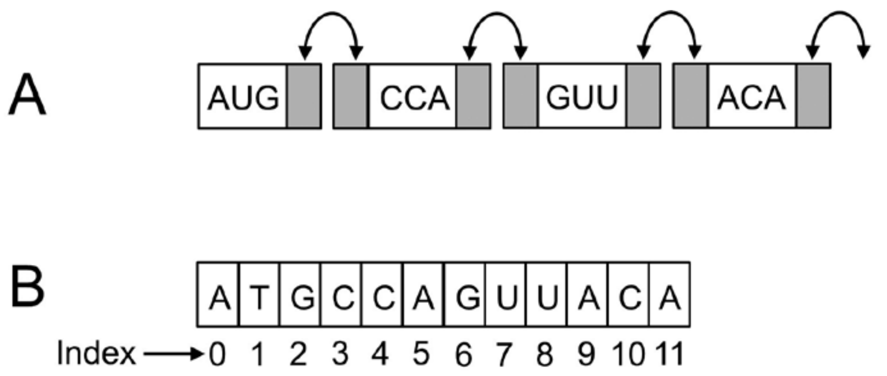


Figure 10. Linked lists and arrays. A. Double linked lists contain data (non-shaded boxes) and links (gray boxes), which point to the preceding and next member of the array. B. Arrays contain data at static locations identified by index values.

ranges of distance and location between the relevant data elements. (In the case of linked lists, however, a more complicated search for the relevant variables must be implemented).

We suggest below that DNA replication and transcription processing, which are used by different codes than those just discussed, are based not on arrays but linked lists. There are subtle differences between these kinds of data structures. For example, in computers the length of an array is established when the array is created (unlike linked lists, which grow and shrink as needed). Remarkably, in cells the same nucleotides are sometimes used by different codes concurrently, each with different kinds of data structures.

Linked List

A linked list is a chain of data and link values. The data part contains the useful information, and the link has the address of the next or previous element. Single-linked lists only point to the address of the next element, whereas double-linked lists include pointers to the next and the preceding data location (Figure 10).

Either an array or linked list could be used for programming purposes. They do differ, however, in internal implementation in ways that affect execution speed of data insertion, deletion, updating, and searching. One difference is that the index value where specific data is located in array lists is generally not known in advance and can change. Unlike arrays, linked lists can automatically grow and shrink dynamically as needed.

To illustrate the difference, candidate CRMs that could interact with the proximal module to regulate a gene are separated by distances that can vary (Figure 5). Finding the activated CRM requires a search for relevant data symbols whose positions are not defined by unique index values. An additional complication is that the regions of the CRM that are to bind to the proximal module involve CREs whose positions are not static in three dimension and must also be searched for.

The same reasoning applies when spliceosomes identify variable intron content whose boundary is defined by splicing signals (Rino and Carmo-Fonseca, 2009). The introns are generally not identifiable a priori by fixed index positions and the spliceosome succeeds even if transcription error adds or eliminates nucleotides.

In linked lists, elements of a defined data type (which could be a complex group of different item types) can be added to the end, inserted at any position, modified or removed (for arrays also, but that requires more processing effort). In addition, another linked list can be added on to another at any position. One disadvantage, of course, is that more effort is required to find a specific value compared to when its indexed location is known in advance.

In RNA, the four nucleotides {A, C, G, U} are attached to riboses (and deoxyriboses for DNA), which are held together along the backbone by phosphate groups (Figure 11).

Analogous to linked lists, nucleotides can easily be added, removed, or inserted simply by breaking and reattaching "address pointers," here phosphate bonds. This is an excellent description of what happens when DNA chains replicate one base after the other, RNA is transcribed, introns are removed, exons are spliced together, and chromosome crossover occurs. Absolute index values per se are generally not relevant for the logic processing, unlike for arrays.

We summarize in Table I some of the built-in methods available to linked lists, using the Java language (https://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html) and include some examples from cells.

In many cases, the processing could be defined in terms of linked lists and/or arrays. Let us recall miRNAs and take into account the concept of sublists, or relative indices, mentioned in Table I. In processing step 1, the nucleotides of a candidate miRNA could be assigned to a sequential linked list. In processing step 2, sliding windows 6–10 nucleotides long (representing candidate seed regions) could be fed into a fixed-length array. The values in array position[0] … position[9] would then be systematically tested against possible acceptor variables in mRNAs. Multiple hits are allowed.

Method	Meaning
Add()	Appends an element to the end or inserts at a specific position.
Cells: RNA transcription; some forms of RNA editing can insert codons (Bass, 2002; Nishikura, 2010); removing introns and splicing exons together; replicating DNA; chromosome cross-over.	
Clear()	Removes all of the elements from a list.
Cells: Upon degrading RNA all resources are free to be used for other purposes, unlike for arrays which when empty still consumes computer memory.	
Contains() **Get**() **IndexOf**()	Returns true if this list contains the specified element. Returns the element at the specified position in this list. Returns the index of the first occurrence of the specified element.
To identify introns, a primary transcript is searched to identify where it starts and ends to identify the index values. Intron lengths can vary considerably. Automated algorithms, such as SplicePort (http://spliceport.cbcb.umd.edu/) and Gene-Splice (http://ccb.jhu.edu/software/genesplicer/) reflect the logic used in eukaryote cells to identify splice sites. The same concept is found in DNA in which transposable elements can be removed from genomes using patterns that define where they begin and end (van de Lagemaat, 2005). Other examples include: the initiation codon on mRNA is searched for (and modified) and so is the region on mRNA at which to create polyadenylation tails; patterns on mRNA are also searched for where nucleotide posttranscription modifications are to occur. The CRMs (Figure 5) are of variable distance from each other (e.g., after insertion of transposable elements into DNA) and need to be found. The location of elementary symbols for activated CRMs can also be variable, depending on what TFs are bound and which ligands these TFs contain.	

Table I. Some in-built methods used with linked lists in object oriented programming languages like Java and examples from cell biology.

Variables as a Data Structure?

We mentioned that in programming, arrays and linked lists are used to store **data values**. These can be assigned to a variable. For example, for an employee stored in index position 45 we might have a line of programming such as: SalaryInDollars = SalaryInPesos(45) * 1.4, and there is no ambiguity in how the value assignment occurs, nor in what was assigned to the variable "SalaryInDollars." Sometimes this is also true in cells. The anticodon of a specific tRNA is fixed, and the value of the communicated charged amino acid is exactly specified. But in cells this is not always that straightforward. It would be as if the variable SalaryInDollars could have small physical differences that affect how it interacts with the array positions, leading to significant effects. This issue can also apply to variable assignments that do not involve arrays and linked lists.

Unlike computers, cells often use variants of variables that do not respond identically to the same values. For example, a CRE is like a sensor, a variable that can be assigned values such as "TF_n bound" or "no TF is bound." However, the binding sequence of a particular CRE can vary and therefore respond differently to an identical TF (which itself can provide many values). This can have serious consequences, affecting how fast and long binding occurs, and could even affect the subsequent Boolean logic. (For example, a modified CRE might affect the geometry of the bound TF and thus how it interacts with other factors.)

This suggests a novel technical inspiration for computer scientists and bioinformatic researchers. Instead of

Method	Meaning
RemoveAll()	Removes from the list all occurrences of specific values.
Cells: Examples include tRNA splicing (Trotta et al., 1997) and RNA self-splicing (Cech, 2002) based on secondary or tertiary structure. These rely on discrete structures which can be stored as structured (i.e., multisymbol) values in individual linked list positions, which is a different operation than removing whatever is found between two boundary patterns. This assumes a specific code is to work with the linked list. Note: Gene silencing mechanisms are not the same as physical compacting through physical removal.	
RemoveRange()	Removes the elements whose index is between two specified indices.
Cells: After the index location of intron/exon boundaries are found, the introns can be removed from primary transcripts.	
Set()	Replaces the element at positions that need to be specified with a value.
Cells: Error correction mechanisms use a DNA or RNA template; any process which modifies a DNA nucleotide (like methylation) or RNA codon, including RNA editing (Bass, 2002; Nishikura, 2010).	
SubList()	Sublist data structures are a feature of linked lists and arrays. Logic processing is performed with respect to the sublist and its own indices, for which the first one is assigned an index value 0, the second 1, etc. All operations performed on the sublist are reflected in the original full list.
Cells: The seed region within miRNAs. In addition, many of the examples above rely on first identifying the location of boundaries; what is relevant thereafter are the relative positions.	

Table I (continued)

treating members of a class of CREs as functionally identical or as separate variables—as we have been implying so far—the suggestion here is to develop a fuzzy-logic type technology which permits both variables and values to be processed with variability. Finding cellular **variables** would then also use linked arrays, since the candidate regions and length would be unknown in advance. The imprecision of many bioinformatic software tools to identify regulatory patterns reflects these joint uncertainties.

Here is an example. RNA polymerase and TFs search for DNA (response elements, or sensors) in 100–1000 base-pair regions upstream from the transcription start site and on the same strand. Nucleotide positions are indexed with negative numbers counting back from -1 towards the 5’ direction. The patterns to test are variables that are not always the same in location or details, which is where linked lists become useful. In focused initiation, transcription starts at a single nucleotide or within a narrow region of several nucleotides having sequence motifs such as the TATA box and DPE. In dispersed initiation, there are multiple weak start sites over a broad region of about 50 to 100 nucleotides (Juven-Gershon and Kadonaga, 2010).

This suggestion captures those cases where symbols seem to have both variable and value character. The regulatory portion of genes define variables that need data to know when and where to initiate transcription, but simultaneously RNA polymerase and TFs sometimes also provide variables that need data to know where to attach in the promoter region.

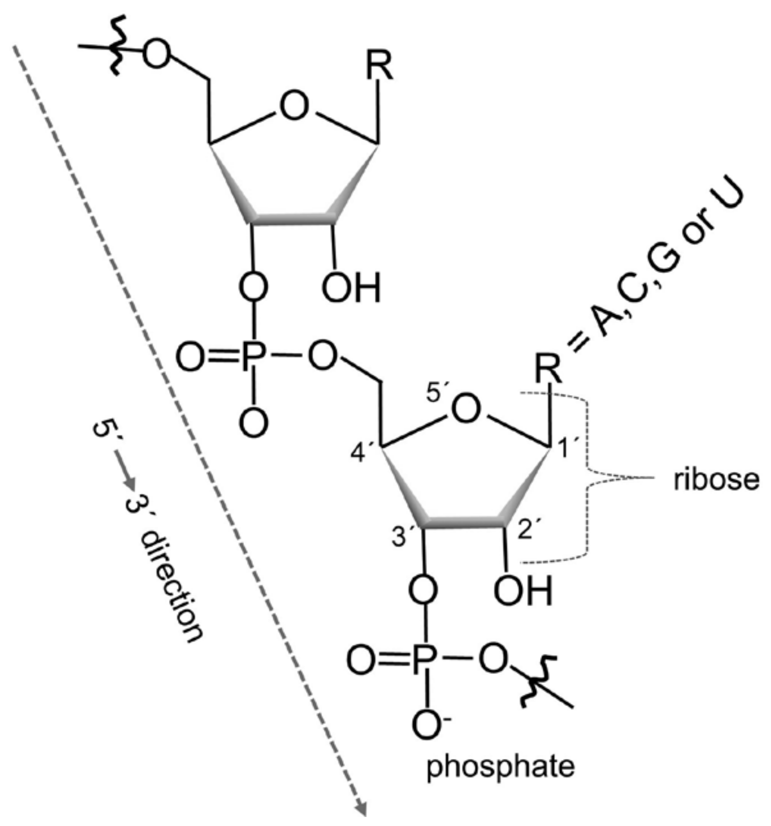


Figure 11. Structure of RNA. The four nucleotides are defined by whether the base A, C, G or U is attached where the R group is shown.

II. Generic Programming Elements

Modern computer languages use some standard constructs to express what is to be done. Often the same logic can be reused, and new values only need to be assigned to the variables. We will discuss the main ones used to implement processing in computer and cellular programming.

Assign Values to Variables

Kirschner and Gerhart noted that information is used by cells to respond to changing circumstances. They wrote, "Two extreme views of information transfer have always existed in biology, the permissive and the instructive. The distinction comes up whenever there is a stimulus and response, or more generally a cause and an effect … Watering a seed provides a stimulus, but it is a permissive input, since no one would assume that the water falling on the seed instructs the seed how to germinate into a plant" (2005, p. 125). We believe their intuition refers to values (provided by the stimulus) and variables (which generate a response upon processing with the assigned value). The cascade of steps to be executed—after, for example, sensing moisture—must already have been prepared and anticipated at the receiving end. The variables patiently wait until activated by informative signals.

Programs and subroutines use variables restricted, as we mentioned, to a relevant data type, to which different values can be assigned every time the program is executed. To illustrate, *price*, *discnt*, *p*, *d*, and *newpri* are variables in this Fortran-like programming code.

```
price = 100                (5)
discnt = 5
call calc1(price, dis-
cnt)
subroutine calc1(p, d)
newpri = p - d
```

Values have been assigned or are calculated. Here *price* and *p* have the same meaning, and two coding conventions are linked by associating a variable from the calling program to one used within the receiving subroutine *calc1*.

How do variables relate to the discussion on symbols, data types, subsets, and operations above? In computer programs, variable names and their values are constructed from one or more fundamental symbols. The **variable** *price* is defined by combining several symbols from the relevant ASCII alphabet and is treated as a unique entity. The symbol combination *ecirp*, however, has not been assigned a meaning in (5) and is not a valid variable in this program. The **value** 100 assigned to price is also comprised of several ASCII symbols, which taken together have a unique meaning, but assigning price = e34/$![makes no sense, being outside the relevant data type. An operation newpri = TRUE / 45 is also not legitimate, not being a valid operation of that datatype.

Through such precise software conventions, programming errors can be avoided and action to perform expressed unambiguously. However, if the semantic meaning of the variables is not known, the ultimate intention and results might never be fully understood. What if the source code is not available at all but only the executable program? By empirically testing variable values, the hidden Boolean logic can still be discerned in principle by the results, an important observation when reflecting on cells.

How does all this work with cells? To understand cellular logic, one must identify four players: the sending coding system; the receiving coding system; and, for both, what the variables are and what provides their values. What is a variable? It is the biological receptor or sensor able to assume alternative values (including a simple "bound"/"not bound" state), which, once activated with a value, leads to a relevant biological response.

Variables are composed of a single symbol or of elementary symbols combined in a unique manner (in computers and cells). Defining variables is necessary to program intention, and cellular variables are identifiable by humans and cellular decoders.

The A site of a ribosome is a **receiver variable** (Figure 2), able to accept as values any of the 64 codons or to be empty. To work properly at the ribosome, not any codon will do. It must only accept the value transferred by a specific **sender variable**, which is associated with the relevant mRNA.

As another example, the location on a template DNA being currently processed by a DNA or RNA polymerase is a **sender variable** whose current value is one of the four nucleotides to be communicated to a polymerase decoder. At the end of the growing chain, part of the polymerase defines a **receiver variable,** which needs to know which nucleotide is to be added (the receiver value) (Figure 3).

The general pattern should now be clear. Special locations on sugars, membranes, or proteins are variables that can accept values (ligands or nothing bound), for example, in the histone code. The enzymes that methylate the appropriate histone residue can have many variables of their own—used to first perform their own internal logic—and then a sender variable is assigned a sending value, the ligand it will transfer. Recall that a chain of sender/receivers can be set up.

The discussion above may have suggested that only a few elementary symbols are used along with a handful of values for variables. Unlike computers, which use only elementary 0/1 "bits" grouped into a relatively modest number of unique ASCII symbols, in cells variables and their values rely on different and more complex alphabets for different codes, using many elementary symbols having distinct geometric and electronic properties.

With computers, hardware design is simpler and more reliable if the variety of elementary symbols (bits) and grouped symbols like ASCII letters are restricted. Many of the cellular codes, however, must support a far more nuanced behavior (recall our comments on fuzzy variables and fuzzy values). A very large number of elementary symbols are used, each having three-dimensional electronic and geometric features (as when portions of amino acids within proteins are combined in TFs). This permits rheostat-like or fuzzy-logic outcomes, which can be fine-tuned dynamically.

To illustrate, not only can different combinations of amino acids define the same kind of TF, but nearby attachments and physical conditions like temperature and salinity can affect the quantitative value that gets interpreted once bound to a CRE. Fine differences in the topology of the same kind of CRE—even those having identical nucleotides—can also lead to quantitative differences upon interacting with a seemingly identical TF. This is important to understand how codes can interact synergistically. They can modify the physical geometry of the compound symbols used by other codes.

Assign a Value to a Constant

Values of variables could change very often during execution of a program, such as the next nucleotide value to be processed by a polymerase. Programs also benefit from using constants, which during a relevant time period should not change. Implicit in cellular logic processing are many constants, such as the temperature, amount of energy provided by an ATP molecule, which ensemble of genes are up- and down-regulated for a cell type, and genomic imprinting (in which certain genes are expressed in a parent-of-origin-specific manner).

Boolean Logic

The ability to use If-THEN-ELSE type logic adds immense value to programming, and to understand cellular logic, one must identity what is the *variable* being tested and what provides its *values*. Between 5% and 10% of protein-coding genes in most organisms encode a TF (values for CREs), and these can have multiple binding domains. Only three kinds of domain are known: cold shock, helix-turn-helix (HTH) type 3, and HTH psq (Walhout et al., 2013, p. 67). Interaction of only portions of a domain with a CRE or other biochemicals define the values (Figure 12).

Example (6) illustrates in programming terms the kinds of logic performed.

```
if (CRE_1 = 'val_1')
{do this}                (6)
else if (CRE_1 =
'val_2') {do something
else}
else if (CRE_1 =
'val_3') {do the follow-
ing}
else {do nothing, or
continue what you are
doing… whatever makes
sense}
```

Checkpoint if-then logic occurs throughout every step of the cell cycle (Shapiro, 2014) checking for genome damage (Ishikawa et al., 2006), nutritional status (Searle et al., 2011), progress of replication (Segurado and Tercero, 2009), DNA replication (Putnam et al., 2009; Nguyen et al., 2010), DNA damage (Huen and Chen, 2010), chromosome alignment on the spindle pole (Nezi and Musacchio, 2009; Musacchio, 2011), spindle orientation (Caydasi et al., 2010), telomere capping (Ciapponi and Cenci,

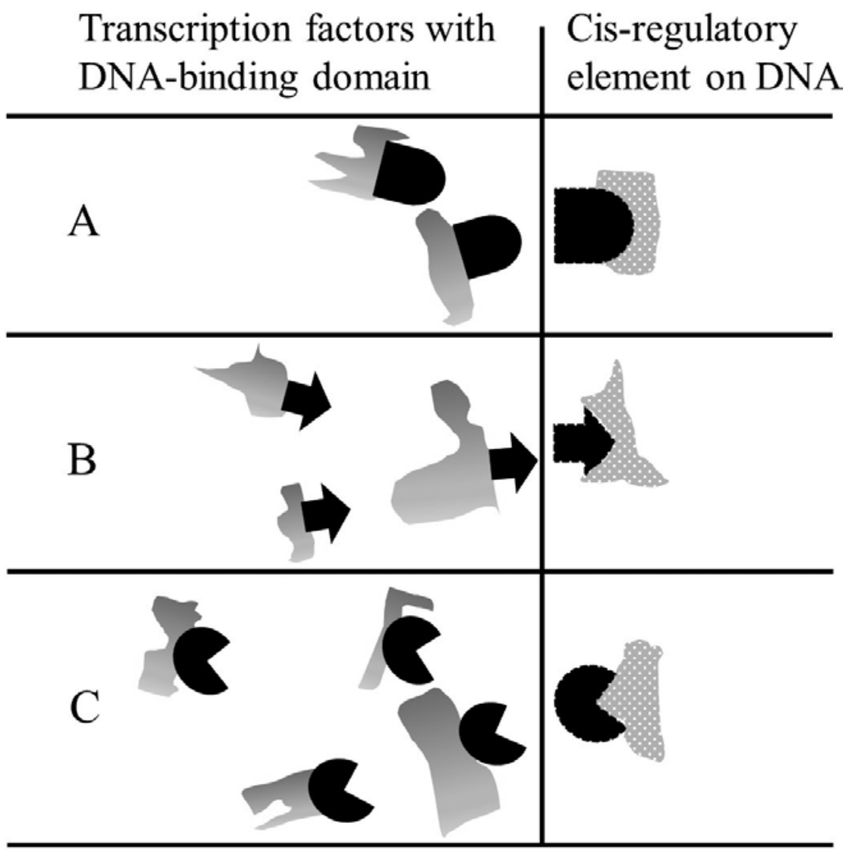

Transcription factors with DNA-binding domain
Cis-regulatory element on DNA
A
B
C

Figure 12. Transcription factors possess DNA-binding domains (solid black), only portions of which provide the values for receiver variables (the appropriate cis-regulatory elements).

2008), cell size (Fang et al., 2006), and whether the cell has accumulated the necessary components needed by the daughter cells (Sabelli et al., 2013).

Errors would lead to serious consequences. Instead of genome repair in response to DNA damage, the if-then logic could lead to programmed cell death (apoptosis) (Tentner et al., 2012; Walsh and Edinger, 2010; Engelberg-Kulka et al., 2009), using some intercell molecules as "death factors" (Holoch and Griffith, 2009) or to a decision to halt the cycle and initiate very sophisticated repairs (Song, 2007).

Iteration

Iteration loops are often used in programming to ensure the correct number of repetitions. An "infinite loop" would consume a computer's—and cell's—resources and must be prevented (Figure 13).

Various repetitive processes occur in cells under the careful regulation of Boolean decisions: many RNA copies are produced from a single gene; many protein copies are made from a single mRNA; many copies of key biochemicals are synthesized, such as amino acids, tRNAs, hormones, ATP, antibodies, etc.; each codon position on mRNAs must be processed; flagella must rotate enough times but not continually; tubulin copies are polymerized to form long microtubules; enough recursive interactions having the right parameters must be run to produce steady-state genetic regulatory circuits; and many copies of each cell type are produced in eukaryotes.

There are many more examples, recognized whenever a cyclic behavior is observed having feedback control. Examination of molecular machines reveals that this is a general principle. Controlling iteration, defining the conditions to use, when to start, and when to terminate, must be implemented simultaneous with the iterating processes. Runaway production would be deadly. Remarkably, this applies not only to the operation of molecular machines but also to the process to create the right number of them also, according to current cellular need. Structuring data into datatypes like arrays and link facilitates the use of iterations in programming.

Control Structures

Programs use techniques to control what is to be done, when, where, how, and how often. In cells, we find many examples. We discussed iteration already. Boolean logic is used with the binding state of cis-regulatory elements (CRE) such as enhancers, silencers, and insulators (Kolovos et al., 2012; Capelson and Corces, 2004) to regulate genes precisely, in a manner unique to each cell lineage (Davidson, 2006). The logic is often very complex. Suites of *cis*-regulatory modules (CRMs) (Figure 5) can regulate multiple genetic loci distributed throughout the genome, establishing network circuits sometimes called "regulons" or "*cis*-regulatory networks" (Dufour et al., 2010).

The combinatorial potential through binding various TFs permits a vast range of regulatory possibilities, able to engage in sophisticated molecular computations (Shapiro and Sternberg, 2005; Davidson and Erwin, 2006). Because the underlying physical interactions are weak, the components can form and dissociate rapidly to permit quick responses to signals received. Complex computations using weak interactions to form novel circuits is also typical of

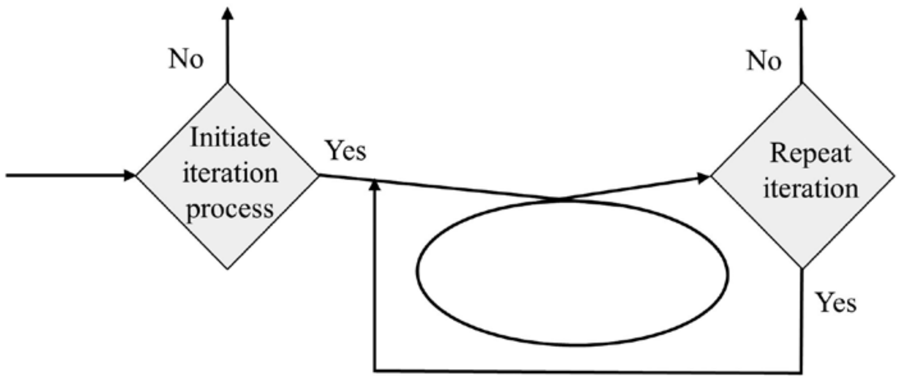


Figure 13. Iteration loops are common in computer and cellular programming. Conditions are tested to determine when to initiate an iterative process and when to repeat or terminate it.

how neurons are wired (London and Hausser, 2005; Sidiropoulou et al., 2006; Markram et al., 2015).

Computer programmers can use "**GoTo**" type commands. Special signals are ubiquitous in cells, which specify where molecular machines and components are to act, i.e., which organelle, subcompartment, or location on a membrane. Causing instructions that are stored elsewhere to be executed goes by names such as functions, methods, procedures, and subroutines in computer programming. In cells, there are many examples, such as activating hox genes to regulate expression of many genes as a modular ensemble and activating key TFs to generate genetic regulatory genetic circuits (Davidson, 2006). Remote processing is often encapsulated in various subcompartments and organelles. We recall that DNA is also present in plasmids, mitochondria, and chloroplasts, not just chromosomes. These decisions also require the use of variables.

Another technique used by computer languages is the idea of "sleep" or "**wait**" for a fixed or variable time period. We find many examples in cells, such as feedback inhibition in enzymatic networks, gene deactivation, and placing the cell cycle on hold.

Other Non-Prescriptive Processing

Most of what happens in computers results from explicit instructions, but our analysis of coded information systems clarifies that additional physical constraints are also always incorporated to ensure the intended outcomes. There are design trade-offs, whether to guide intention as coded messages or in a hard-wired physical manner. A computer example is when printed paper falls into a tray with sides that hold them in place. A considerable amount of cellular success is based on pure physical-chemical factors that have been carefully organized, a topic we discuss in Part 2.

Read and Write

Computer programs read, write, and delete to long-term and short-term memory devices. The codes found in cells must be able to read and write data values. Setting epigenetic tags are examples of medium and long-term write operations, which serve to communicate intended outcomes later. DNA is usually thought of as a fairly permanent source to read data from, but DNA can be added to a genome via CRISPR (Zetsche et al., 2015; Ran et al., 2015; Gen News Highlights, 2015), reverse transcription (e.g., telomerase reverse transcriptase that maintains the telomeres of eukaryotic chromosomes), transfer and acquisition of new genes via integrons coding cassettes (Hall and Collis, 1995), and different lateral gene-transfer mechanisms, including transfer of plasmids, in prokaryotes. Inteins are another mechanism. These are self-splicing portions of proteins with homing endonuclease ability that snip parts of DNA so that a copy of the coding sequence of the intein can be inserted there (Gogarten et al., 2002).

DNA can also be modified in other ways. DNA segments such as transposons can be transferred to other sites on the genome, and "shufflons" can invert sections of DNA, for example, to replace part of a coding strand with its complementary strand to create modified proteins (Tam et al., 2005; Komano, 1999).

Multiprocessing and Threading

Modern computer hardware and software designs can parallelize computations, permitting multiple tasks to be carried out simultaneously. This is common in cells, such as in the parallel production of ATP from many mitochondria; translation of several identical mRNAs in parallel (several ribosomes can also translate the same mRNA simultaneously), transcription of multiple copies of the same gene, the existence of many cells of the same kind, and the presence of multiple copies of the same subcompartments and organelles.

Reuse of Modules

In good software design, the same general-purpose modules, methods, and procedures are often reused. A common approach is to separate identical portions of coding into smaller modules that can be invoked from within overarching modules. This modularity is found also in cells. As Kirschner and Gerhart pointed out (2005, p. 137), "The same pathways are used over and over again within the same organism for different purposes. Thus, they must be modified slightly to interact with a variety of processes and to work in different environments and cell types." They describe the interactions as "weak linkages," which we recognize as simply variables or parameters used to link subprocesses in different manners.

Interchangeable Libraries

In addition to invoking subroutines, sections of computer code such as classes are often imported from a library. Similarly, prokaryotes in particular exchange genetic material through horizontal (lateral) gene transfer (Thomas and Nielsen, 2005; Ochman et al., 2000; Koonin, et al., 2001), whereby genes, plasmids, and so called "islands" encoding specialized adaptive functions are exchanged (Dobrindt et al., 2004). This permits a huge amount of coding to be distributed in the environment and put to use rapidly when the need arises, facilitating adaptability. This is a form of open systems design. Genetic material can also be transferred into eukaryotes through vectors such as viruses.

III. File formatting

Shapiro and Sternberg (2005) drew attention to the parallels between computer file formatting and data storage in cells:

> The explicit parallel with electronic data systems indicates that the genomic storage medium has to be marked, or formatted, with generic signals so that operational hardware can locate and process the stored information. (Shapiro and Sternberg, 2005)

Data storage can be organized physically in computer and cell technologies using principles such as sectors, disk partitioning, and data segments, discussed in Part 2. On top of this infrastructure, software programs organize different data using file formatting. A program that interacts with specially structured file data must be able to access it correctly, even though the location of the content could be scattered all over the physical medium. DNA, RNA, and proteins are used as read/write/delete storage devices and need to be properly formatted so the corresponding "reader" will work.

The metadata contained in a computer file header can be stored at the start, end, or other areas of the file. Likewise, in DNA, RNA, and proteins formatting instructions need not be found in only one location. Given the existence of multiple codes, DNA "files" are formatted for use in different manners, depending on the program being used. The various ways DNA are packed, such as by nucleosomes, determine which genes can be processed. Preparing portions of DNA for processing by DNA polymerase (to identify the starting and end points, open and unwind the strands, remove bound histones, etc.) is very different from the formatting details—which occur in three dimensions—for RNA polymerase. The programs that perform DNA error corrections also require their own formatting rules.

Epigenetic tags are often used to identify what data to process and how. Adding and removing these ligands from DNA, RNA, and proteins is an example of preparing files for processing and must be carefully regulated. Histone modifications define which portions of DNA can be processed. Over a hundred posttranscription modifications have been identified in all three major RNA species (tRNA, mRNA and rRNA), as well as in other families of RNA such as snRNA (Cantara et al., 2011). Examples of formatting specifications in DNA include the use of methylation and demethylation (Bird, 2002; Paszkowski and Whitham, 2001), binding of TFs (Cheng et al., 2012; Davidson, 2006), and rules to identify exons (Harrow et al., 2009).

Individual eukaryote mRNAs are formatted as individual files with beginning and ending metadata in the form of 5' capping and 3' polyadenylation, attached miRNAs, and so on. This is necessary to ensure the ribosome program will work properly. Different sets of formatting rules are necessary for different programs such as separation of introns and exons by the spliceosome or to degrade RNA.

Formatting on proteins is common. Posttranslation modifications (PTM) include methylation, phosphorylation, acetylation, ubiquitylation, glycosylation, and sumoylation (Strahl and Allis, 2000; Jenuwein and Allis, 2001). Structural three-dimensional recognition features, generated with alpha coils, beta sheets, disulfide bonds, hydrophobic patches, and other features also ensure correct formatting of proteins. In cells, all this is precisely regulated, often down to the atomic level. Reversible phosphorylation, the most widespread PTM, occurs on the correct atom of a serine, threonine, or tyrosine residue to form phosphomonoesters or on histidine, arginine, and lysine residues to form phosphoramidates (Ciesła et al., 2011), all according to the particular code involved. Recalling the existence of signal cascades and enzymatic networks, proteins are also carriers of data values that get processed by other sensors (variables to be assigned values). DNA and RNA are not the only information carriers in cells. A modification on a TF can become a data setting to be used by the receiving portion of a second TF. For these reasons we see that proteins can be formatted and classified into different "file types."

Copies of tagged proteins, RNA, and DNA (like nucleosomes) can be inherited by somatic daughter cells, and sometimes the tag is removed from the daughter cell, generating an empty or partially empty "file" that can be written to. In the same way that a program like Excel cannot process a jpg file, each of the cellular information "readers" process only the data specially formatted for it.

Compressed Archival

DNA is compressed and protected for future use by winding sections of ~ 147 base pairs around a core of 8 positively charged histone proteins into nucleosomes, and then further compacting the nucleosomes into higher order chromatin structures complexed with protein and RNA (Jenuwein and Allis, 2001). The portions of DNA that need to be expressed must be unpacked and reformatted properly. The cellular goal is to save physical space and protect the medium from degradation. Bacteria also quickly lose DNA not immediately needed (and can regain genes via lateral gene transfer), which saves precious raw materials and energy. Computer analogies of the principle include programs like zip and the export of tables by a database management into a single export file, all or parts of which can be retrieved and properly structured for use later. However, computers use algorithms that recode the original content using fewer bits, a principle not known in cells. Inspired by cellular compression, which transforms essentially linear DNA into three-dimensional storage, engineers might consider designing mass storage devices to also store data that cannot be used immediately as is but, like packed DNA, could be reopened when needed.

Summary and Discussion

Recognizing cells as information processing devices is the proper way to understand their holistic intent and design. In fact, Gatlin (1972, p. I) defined life as an "information processing system," and Britten (Britten, 2003, p. 82) pointed out, "We cannot start with DNA and grow a cell because there must be an adequate initial state of a cell with a vast multitude of details under control." We mentioned above that cellular information is partially distributed hierarchically and recognize that there are many carriers in the lower, embedded levels. An organ consists of many cells, each of which contains many mitochondria, and so on. In large populations of prokaryotes, the logic processing is distribution over many interacting species to form a viable ecology, whereas in complex eukaryotes considerably more is concentrated within the individual organism. In virtually all biochemical processes, one sees strong regulation unless the process is malfunctioning, as in cancerous growth or viral infection. In other words, there are always sophisticated rules for when to begin and countermeasures that prevent runaway processing.

Regulation is best designed and interpreted using purely formal rules, a key feature of software engineering. If, for example, a metabolic chain requires feedback control to a preceding enzymatic reaction, this can be analyzed and expressed symbolically, along with the mathematical specifications and control rules. To instantiate the requirements, a physically viable solution then needs to be implemented. No rational engineer or programmer would think of developing programs by letting rules and their implementation pop into existence randomly without any conceptual guidance.

We saw that conceptual software elements such as iteration and control structures are developed on top of data types—each with their unique properties—organized into variables, arrays, and linked lists and all this using well-defined file formatting to facilitate processing by molecular machines. Many independent codes found in cells make use of these principles. It is hard to overstate how important variables are in cellular processes to permit regulation and maximum adaptability. The location, timing, and amount of transcription by RNA polymerase is defined by CREs (promoters, enhancers, silencers, insulators; Kolovos et al., 2012) and termination by terminator sequences (Ishihama, 2000).

It would require many volumes to describe in detail the formal control structures used by other cellular activities, such as homologous chromosome crossing-over, VDJ recombination in the immune system, nonhomologous end-joining (NHEJ) of broken DNA ends, DNA transposons (self-insertion, excision), telomerase extension, chromosome segregation, DNA compaction, binding sites affecting DNA spatial organization into transcription factories in the nucleus, signals for error correction and damage repair, and the multitude of other cellular processes.

There is considerable evidence that damage through random changes is actively hindered in cells, such as a bias for many retrotransposons to insert upstream of transcription initiation sites (Shapiro and Sternberg, 2005), which prevents damage to coding sequences and enhances the potential for a constructive regulatory change. Very often the regulatory logic makes sense to humans skilled on symbol logic, but the details are different across taxa and did not originate from a common ancestor. An example is the signal used in E. coli to repress catabolism (the CRP palindromic binding site for the CRP-cAMP complex), which is unrelated to that found in Bacillus subtilis (CRE element recognized by protein CcpA) (Miwa et al., 2000).

Coded Systems Can Interact

Although the various codes operate independently in cells, they can collaborate to ensure a fine-tuned outcome. We mentioned epigenetic codes, which modify gene expressions, and another code based on TFs bound to CREs, which also regulate gene expression. But in addition, a different code based on adding and removing ligands—especially phosphate groups—modify the TFs themselves (Shapiro 2006). Furthermore, TF half-lives are also regulated by the NEnd code. Gene expression is further affected by other codes which use various classes of RNAs (siRNA, snoRNA, miRNA, etc.) that modify chromatin accessibility, transcription initiation, transcription elongation, RNA processing, RNA stability, and mRNA translation (Mattick and Makunin, 2006; Taft et al., 2010; Storz and Wassarman, 2005).

By integrating multiple codes, cells become highly responsive to what is going on throughout the entire cell and their external environment. The design requirements would be overwhelming for humans. The same stretch of DNA can be used as variables for some codes (e.g., CREs, methylation binding locations, and after transcription to locate regions on mRNA for miRNA binding and to specify intron/exon boundary locations) while simultaneously providing the data values for other codes (e.g., as codons after transcription and as a template for new DNA copies).

These requirements demand formal specifications to satisfy all requirements and to define what is to be done by each code. Synonymous coding from the point of view of the genetic code must identify protein sequences while simultaneously controlling translation rates within regions of mRNA. The DNA-to-RNA conversion code during transcription also needs to control stalling of mRNA precursors for spliceosomes for purposes of siRNA accumulation as part of a host's defenses to damaging transposons (Dumesic et al., 2013).

Collaboration between coding systems is sometimes linked directly. The histone modifications, which involve over 100 protein readers, writers, and erasers (Carey, 2012, p. 72, 224), sometimes develop protein complexes that include the enzymes that methylate CpG motifs on DNA (DNMT3A and DNMT3B) in the same region the histone is located (Carey, 2012, pp. 73, 89–90). This is another example of instantiation using adaptor molecules.

The reverse is also true. The DNA methylation code can affect the histone code in a synergistic manner. Methylation attracts more repressive histone modifying enzymes (Carey, 2012, pp. 224–226). Similarly, long ncRNAs locate near imprinted genes (which identify whether coming from the mother or father), and these can recruit epigenetic enzymes such as G9a or EZH2, which put a methyl tag on lysines K9 and K27of histone H3 (a second code) to enhance the imprinting (Ikegami et al., 2009). To complicate the picture, long ncRNAs can increase or decrease expression of target genes for reasons not understood.

The miRNA code also interacts with enzymes involved in epigenetic codes by regulating their effective concentration (Carey, 2012, pp. 231–232).

Stem cells express a very different set of proteins than differentiated lineages. Not only are different genes deactivated by blocking TFs bound in the cis-element region, but also a different set of miRNAs are switched on (a second code) to help identify and degrade the mRNAs no longer needed by that class of cells (Pauli et al., 2011).

Chemotaxis (ability to swim toward nutrients and away from noxious stimuli) uses two codes in E. coli to respond to more than fifty substances. In the first one, there are four kinds of receptors on the membrane that respond to the environment by phosphorylating the communication protein CheY, which can modify the direction of rotation of the flagellar motor through binding at certain locations. A second code affects the four kinds of receptors themselves by adding and removing methyl groups to any of eight different sites per receptor. The receptors are grouped into triplets on the membrane, so the number of possible methylation states is astronomically large. The net outcome of these two coded processes is to permit the bacteria to "in effect perform calculus" (Bray, 2009, p. 94). It is not the absolute concentration of external stimulant that determines the decision to change direction of movement, but rather a large change in the relative concentration (Bray, 2009, pp. 89–97).

In Part 2, we flesh out our understanding of cells as holistic entries whose hardware components must also be taken into account in addition to the interacting codes. It is wrong to think DNA provides detailed instructions on how to assemble an organism. Oyama (2002) pointed out that "a gene initiates a sequence of events only if one chooses to begin analysis at that point: it occupies no privileged energetic position outside the flux of physical interactions" (p. 40) and that "gene transcription and translation in no way represent instructions for building a functioning body" (p. 69). She correctly mentioned that the interactions needed to define organisms are inherited as already functioning cells and in a similar environmental context as the parent (pp. 17–18, 26, 43–49, 77).

Dynamic Nature of Cellular Control

The location of data in computer memory is rearranged in controlled manners and address pointers are used to identify the location of data. For cells this is also true, but the process is more sophisticated. A TF can search for a CRE in three-dimensional space and is robust to physical degradation of its target through mutations. Unlike a computer pointer to a single address, in cells *n* identical TFs or other signals can point to multiple locations to activate an ensemble of process-related genes. In computers, a memory address is usually referenced directly, whereas in cells often a linked chain of pointers referencing other pointers lead to the sites to be activated, which permit refinements, including fuzzy logic (Kosko and Isaka, 1993; http://zadeh.cs.berkeley.edu) to be integrated at every step.

Analog Computers

We have not mentioned principles from the less-known analog computers in this introduction to logic processing in cells. We only wish to point out here that the wide diversity in sensors responding to signals can produce a rheostat-like response (i.e., a continuum of response). Software designed for digital computers would process this kind of logic by defining ranges of values for these variables and program the appropriate behavior for each range. This relates to our suggestion above that computer scientists consider using fuzzy variable and fuzzy values, being a principle cells use.

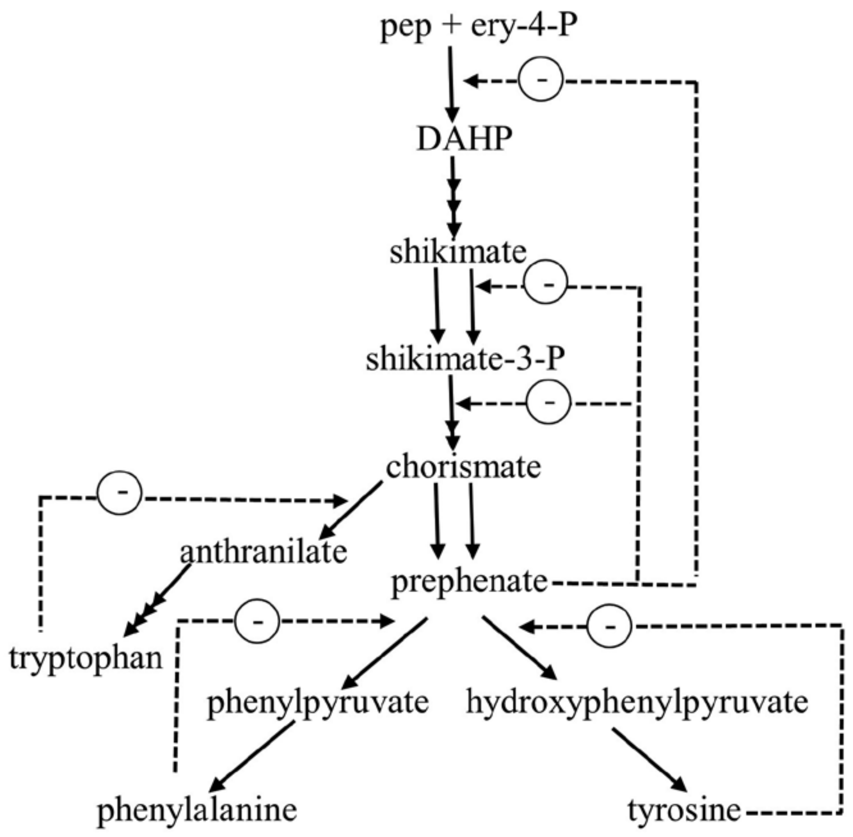


Figure 14. Enzyme chain including feedback in aromatic amino acid synthesis (Fell, 1997, p. 209).

Neo-Darwinism Fails to Explain the Origin of Logic Processing

In *The Plausibility of Life*, we read, "The architecture of cells is achieved without an architect. No central regulation is discernible. Cells are in fact capable of many structures; many are chameleons that change their structure in response to circumstances" (Kirchner and Gerhart, 2005, p. 148). It is correct that there is no set of instructions on DNA that specify the detailed order in which events are to unfold, but this does not deny an architect; in fact, it indicates a creator who designed for adaptability to changing circumstances (Truman, 2015). As mentioned above, a virtually unlimited variety of responses can be executed by using enough variables and their values. Adaptability is found everywhere in biology, not only within cells. Gilbert (2003) provides several examples of dramatic polyphenism, or open systems adaptability, such as sex determination of blue-headed wrasse larva depending upon the presence of other males or females nearby; diet in caterpillars, which enables them to change their morphology to camouflage themselves according to season when born; and predator-secreted chemicals.

Cellular process must be initiated and stopped. Runaway execution would rarely if ever be acceptable, but why should the termination rules develop in advance of these thousands of formal logic-guided processes? Which evolved first, the process or the means to turn it off? Natural processes cannot look ahead to plan complex solutions to make cells and entire organisms adaptable. Gene regulatory networks, signal cascades, metabolic networks (Figure 14), and the operation of molecular machines are regulated at many levels using programming constructs recognizable by human designers.

There is no analogy in inanimate matter of codes being used to express an intended result to ensure continued system integrity. This will become clear after examining in the next paper how extraordinarily complex the molecular machines are which are needed to implement the code specifications. In Figure 15, we clarify the principle, which is not found anywhere in inanimate nature.

The intuition is that a system with complex internal components will be repetitively confronted with a decision that can be freely made, independent of chemical or physical compulsion. For each iteration a particular choice between alternative paths is correct to facilitate the survival of the system (plus the decision-making apparatus), based on current circumstances.

The cell is full of this decision principle, such as where to initiate and stop transcription, which amino acid to add next to a growing protein, and where a restriction enzyme should cut.

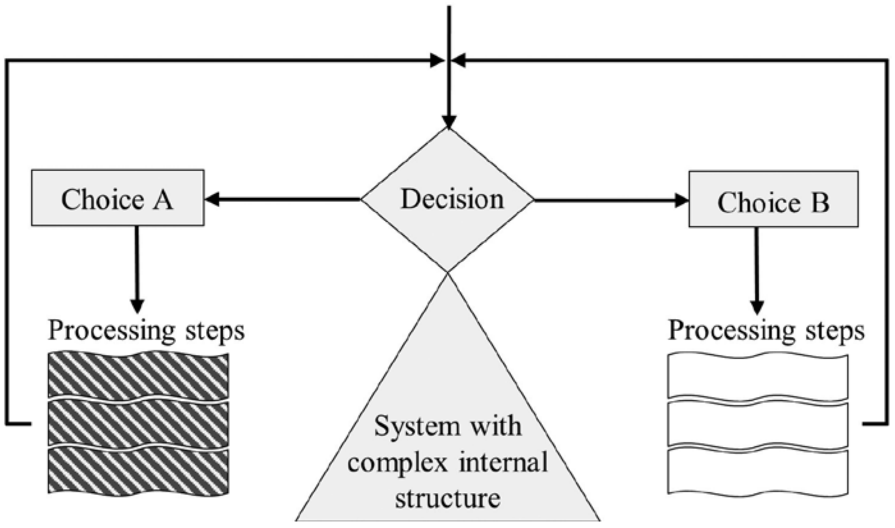


Figure 15. In inanimate nature we find no examples of systems with complex internal structure repetitively facing a contingent decision and then making the correct choice for each iteration based on which outcome supports survival of the system during that iteration.

We do not find any examples in inanimate nature, even though this is but a minimalist requirement. We are not demanding this occurs reliably a huge number of times (like millions of correct peptide bonds during a cell's lifetime) or that it be able to manufacture all its key components, or that the entire apparatus be reliably replicated for many generations. We ask only for examples showing the basic concept is found in some elementary way in inanimate nature. Otherwise, no evolutionary theory is justified in simply assuming grotesquely more complex cells arose in the absence of intelligent guidance. This is essentially asking about the origin of information, like the sequence of codons specifying the correct protein chains.

We know from computer technologies how important proofreading and error correction (parity bit rules, etc.) are during data storage and transmission. In cells, this is far more important, given the many examples of iteration and millions of decisions per second involved. We will consider just one code, the genetic code, to illustrate the need for extreme reliability. If the multiple copies of mRNA and their translation products were error prone, this would lead to error catastrophe during the cell's lifetime. Each new batch of flawed proteins and RNA would lead to ever more defective transcription factors, RNA polymerases, ribosomes, spliceosomes, error-correcting enzyme complexes and posttranslation machines, thereby producing ever more defective proteins and RNA the next time around. The same, of course, is also true about all the components inherited by daughter cells, in particular flawed DNA copies.

Is this a serious problem? The probability that an amino acid will be translated correctly depends on many factors, but suppose that in the distant evolutionary past, before elaborate error-correcting molecular machines existed, natural processes had somehow miraculously reached a state where each of the twenty amino acids was translated correctly with an average probability of 0.80 and that proteins back then were on average only 200 residues long. The chance of obtaining a correctly translated protein would be $(0.8)^{200} = 4\text{x}10^{-20}$.

One recent study of 40 proteins examined in HeLa cells concluded that the lowest number of copies per cell at a given time was for the oncogene FOS (6000 copies), and the most abundant was vimentin (20 million copies) (Zeiler et al., 2012). An ancient primitive organism would not have so many copies. We would not expect to get even one correctly translated protein but a sea of hopelessly flawed, misfolded, and destructively interacting ones (for a more exact analysis see Part 2). Even if the cell could somehow recognize and degrade mistranslated ones (somehow using molecular machines that themselves are hopelessly corrupted), the energy cost to produce enough attempts to generate thousands of necessary good copies would be prohibitive.

What is the reality in all cells studied? Success rates on the order of "only" 0.8 per monomer copied? Many processes recognize and correct errors, such as when DNA is replicated or tRNAs are charged. In exonuclease proofreading during DNA replication, a mismatched duplex is identified and the most recently incorporated nucleotides removed and replaced, eliminating about 99.9% of accidental misincorporations from the nascent strand (Kunkel Bebenek, 2000; Ibarra et al., 2009). A second mechanism, postreplication mismatch repair, then corrects about 99% of those misincorporations that escape exonuclease proofreading (Modrich and Lahue, 1996; Kunkel and Erie, 2005). There is also a molecular machine to repair double-strand (DS) breaks (Brissett and Doherty, 2009).

The other codes must also be highly accurate. TFs could bind to a multitude of wrong locations on DNA; epigenetic tags on histones or DNA must be carefully controlled; flawed signal sequences would cause proteins to be secreted improperly; etc. Furthermore, the coded variables and variable values must be replicated accurately over many generations, not just the organism's lifetime. A distinct combination of millions of methyl tags on DNA cytosines is unique to each cell type and needs to be replicated in daughter cells (Carey. 2012, p. 60) as shown in Figure 16.

We do not know how accurately the methylation pattern per CpG must be replicated for the daughter lineages to still work, but suppose it would be enough if "merely" 1/10,000 need to be correct (i.e., 99.99% error rate would not matter, the resulting pattern would still work). Per replication and one million CpGs, a successful outcome would only occur $4x10^{-44}$ of the time (i.e., $0.9999^{1000000}$), even given such generous constraints. In other words, getting an acceptable copy will not occur even if only 1/10,000 tag positions need be correct on average. We conclude that evolving this new function cannot start crude and be refined by random mutations, since natural selection would have nothing functional or consistent to work on.

Many researchers, especially those of a neo-Darwinian persuasion, continue to downplay the evidence for deliberate planning found in cells, preferring to hold on to the myth that most the genome is junk instead of facing the reality of multiple codes and an overarching systems design. The origin of complex features is assumed to result from random mutations followed by natural selection without recognizing or addressing the origin of formal logic processing (Dawkins, 1996). Absent informational guidance, the only alternative is to believe in a series of naturalist miracles, such as an initial functional genetic apparatus followed by many more miracles including a regulated energy source (ATP molecules) and requirements such as being able to distribute chromosomes and other components to daughter cells.

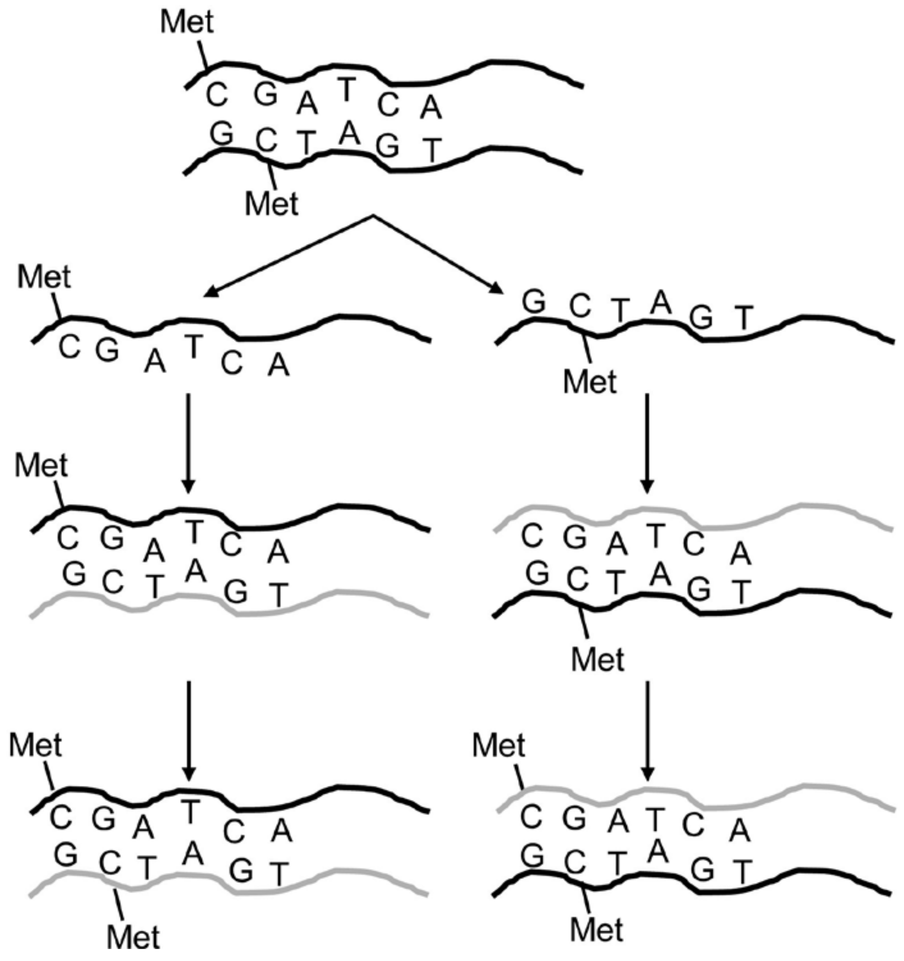


Figure 16. DNA methylation patterns need to be replicated in daughter cells during somatic cell division. After each DNA strand is separated and the second strand copied, the DNMT1 enzyme searches for CpG motifs and transfers a methyl (Met) group to the new strand where needed. This results in two new copies carrying the original methylation pattern.

Is this also a probabilistic nightmare? There are $2^{(46 \times 2)} = 5x10^{27}$ ways to distribute human chromosomes during mitotic cell division (Page and Hieter, 1999), of which only one is correct. There is a better chance to guess two people correctly in a row out of everyone who ever lived. And these odds need to be overcome by every surviving cell every generation, so once again error cascade is the natural consequence until the process is close to flawless. Natural selection is only relevant once the system has attained miracle-level perfection.

In general, whenever we come across the terms "convergent evolution," "genetic piracy," or "co-optatation," we will discover a failure of neo-Darwinian theory and in all likelihood further evidence that logic processing elements are being deliberately reused in unrelated organisms. For many years the very small amount of data available was misused (and continues to be) to claim that a gene expressed as part of the same or similar processes reveals common ancestry. In the words of Striedter (Striedter, 2003, p. 287), "Unfortunately, we now know that most genes are expressed in several different locations and that many homologies based on the expression patterns of single genes have turned out to be controversial, to say the least."

Until one accepts that cells are designed logic processors, much data will continue to be misunderstood. The same transcription factor or the same cis-factor pattern could be reused for biologically unrelated purposes across the biosphere. In programming, we also find software elements such as "for (int *i* = 0; i < *myList.size*(); *i*++)" in many programs, but this does not imply the programs are related in any manner. The *i* and *myList* could represent totally different things.

In discussing the Pax-6 gene found in vertebrates, Drosophila, squid, and even flatworms, Willmer provides an example: "Although this could imply a common starting point for all eyes, it is more likely an example of the universality of positional and pattern-forming determination systems in animals. Note also that while Pax-6 in vertebrates is homologous to the Drosophila gene eyeless, other genes related to eye formation in vertebrates match bizarrely with genes involved in appendage formation and with muscle formation in fruit flies; and that Pax-6 also regulates the unrelated phenomenon of nasal placode formation in vertebrates" (Willmer, 2003, p. 38).

Premature evolutionary speculation, treated and repeated for decades as proven scientific fact, is being increasingly corrected. Discussing the claim that the gene *engrailed*, which is expressed in both *Drosophila* and chordate metameres, proves that segmentation of body parts goes back 500 million years ago to a common ancestor, Willmer explained what more data now actually reveals: "This now seems an overinterpretation. Although homeobox proteins function as transcription factors for other genes, the genes they regulate are often quite unrelated to segmentation. Furthermore, this same Hox sequence appears in a far greater range of animals, including unsegmented nematodes and echinoderms" (Willmer, 2003, p. 39). After providing other examples, Willner then arrives at the correct intuition: "The similarity of genes … may lie in processes rather than in real homology" (p. 40).

Scientific Guidance through the Design Presupposition

The NIH Roadmap Epigenomics Consortium is collecting a huge database with DNA accessibility, RNA expression, histone modification, and DNA methylation patterns for 111 human reference epigenomes (Kundaje et al., 2015). One goal is to identify regulatory modules that arise during cell lineage specification and differentiation. This is representative of the general direction modern cellular research is beginning to take, where it has become indispensable to apply principles from symbolic logic processing to understand in detail the design of cells. Speculative neo-Darwinism is at best post-facto storytelling; it provides no insights into the big, interesting biological questions.

The view that cells were deliberately designed to be robust and adaptable for long-term viability and interactivity, along with the insights of logic processing principles from computer programming, stimulates many fruitful ideas to guide future ideas that do not arise from the evolutionary worldview. Freed from the shackles of possible biological functions being constrained to what a primitive common ancestor initially provided and a limitation on mutational accidents to generate nontrivial novelty, we suggest how our paradigm provides value to guide future research.

1. Cells will be found to be more adaptable than suspected to situations not encountered before, and when the mechanisms are researched, we will find the adaptive logic has coding aspects, meaning the variables were already there and able to process additional values. Asking how one would formally design an optimized outcome, independent of any misguided prejudice from common ancestry constraints, should help identify new cellular control processes. (Post-facto claims for unexpected "convergence" is scientifically worthless and contradicts neo-Darwinian expectations.)
2. Many more forms of complex regulation remain to be discovered than suspected. No iterative process will be found that lacks a formal set of rules on how to initiate and terminate (unless malfunctioning). Whenever it would make sense for the concentration and distribution of biomolecules to vary, we predict evidence will be found this has been implemented in a context-appropriate fashion.
3. Given our conviction that cells were designed to function as holistic and integrate entities, we predict ever more discoveries of interconnectivity between codes so that inputs throughout the entire cell and ecosystem can be taken into account to regulate processes optimally. We expect much will become clear only as the optimization trade-offs are understood and that quantitative analysis will reveal there could not have been nearly enough evolutionary trial-and-error attempts to explore and fine-tune these optimized trade-offs.
4. More quality control checks will be discovered at key processing points. Researchers should search for error checks/correction during transcription to RNA and other key interfaces. Considering the value to cells of recycling valuable raw materials of every kind, we anticipate novel

discoveries designed to ensure this. Conversely, if substances (like cyclic RNAs) are found to be long-lived, we suggest the Creator had a biological reason.

5. We expect that when important alternative pathways are available, the overall optimal one under those circumstances is selected unless clearly malfunctioning. As an example, whether to attempt error correction or initiate apoptosis is a significant decision for cells based on complex cost/benefit/risk tradeoffs. We expect a careful quantitative application of decision theory principles—including Bayesian statistics—to reveal that the outcome selected is overall rational.
6. For every difficult step creating a critical potential processing bottleneck, mechanisms will be found that resolve these, in the same way that we expect that an enzyme will be found to catalyze all key biochemical reactions impacting the survival of a cell. We also anticipate that variants of current enzymes and processes can easily be generated when it makes sense. This is based on our view that general-purpose solutions were often designed, which like good open systems design, are adaptable. Optimized adaptability has nothing to do with the naturalist assumptions going under the label evolution.
7. Since we believe organisms were created optimally (with the goal of filling the earth's ecosystems) but have accumulated errors over time, we will discover residual evidence for functioning solutions in the past, at the cellular or higher level, which do not work as well as before, especially for organisms that have undergone population-size bottlenecks. Applying design reasoning to describe how ideal solutions would work will help us understand how things might have worked before.
8. We will discover multidimensional forms of data storage and retrieval not known for computers. These will be sophisticated beyond anything a naturalist would dare predict. We anticipate the existence of extraordinary code-based methods to store, retrieve, index, network, and consolidate in fuzzy logic and other mathematical forms all kinds of multimedia data (smell, vision, taste, sound, tactile memories, reasoning chains, numbers, facts, etc.) in ensembles of brain cells. We dare predict human minds will be found to be able to interact with these codes in read/write fashion to actively guide queries in a parallel processing fashion.

Acknowledgments

We wish to thank Dr. Peer Terborg and Dr. Niko Winkler for many fruitful discussions. We thank Dr. Michael Heisig for considerable help obtaining much of the literature cited here, Wort und Wissen in Germany for many conferences and opportunities to discuss these topics, Creation Ministries International for years of fruitful collaboration in this area, and Jonathan Bartlett for a detailed review of an earlier draft.

References

André, A., H. Kaltner, J.C. Manning, P.V. Murphy, and H.J. Gabius. 2015. Lectins: getting familiar with translators of the sugar code. *Molecules* 20:1788–1823.

Aricescu, A.R., and E.Y. Jones. 2007. Immunoglobulin superfamily cell adhesion molecules: zippers and signals. *Current Opinion in Cell Biology* 19:543–550.

Ashcroft, S.J. 1997. Intracellular second messengers. *Advances in Experimental Medicine and Biology*. 426:73–80.

Audit, B., and C.A. Ouzonis. 2003. From genes to genomes: universal scale-invariant properties of microbial chromosome organization. *Journal of Molecular Biology* 332:617–633.

Barbieri, M. 2003. *The Organic Codes. An Introduction to Semantic Biology*. Cambridge University Press, Cambridge, UK.

Barbieri, M. (ed.). 2009. *The Codes of Life: The Rules of Macroevolution*. Springer. (See especially chapter 8 by Mario Gimona.)

Barash, Y., et al. 2010. Deciphering the splicing code. *Nature* 465:53–59.

Bass, B.L. 2002. RNA editing by adenosine deaminases that act on RNA. *Annual Review Biochemistry* 71:817–846.

Bernardi, G. 1990. Le génomes des vertébrés: organization, fonction et evolution. *Biofutur* 94:43–46.

Bird, A. 2002. DNA methylation patterns and epigenetic memory. *Genes and Development* 16:6–21.

Bogdan, C. 2001. Nitric oxide and the regulation of gene expression. *Trends in Cell Biology*. 11:66–75.

Bonifacino, J.S., and B.S. Glick. 2004. The mechanisms of vesicle budding and fusion. *Cell* 116:153–166.

Bray D., and T. Duke. 2004. Conformational spread: the propagation of allosteric states in large multiprotein complexes. *Annual Review of Biophysics and Biomolecular Structure* 33:53–73.

Bray, D. 2009. *Wetware: A Computer in Every Living Cell*. Yale University Press, New Haven, CT.

Briscoe, J., A. Pierani, T.M. Jessell, and J. Ericson. 2000. A homeodomain protein code specifies progenitor cell identity and neuronal fate in the ventral neural tube. *Cell* 101:435–445.

Brissett, N.C., and A.J. Doherty. 2009. Repairing DNA double-strand breaks by the prokaryotic non-homologous end-joining pathway. *Biochemical Society Transactions* 37:539–545.

Britten, R.J. 2003. Only details determine. In Müller, G.B., and S.A. Newmann (editors), *Origination of Organismal Form: Beyond the Gene in Development and Evolutionary Biology*, pp. 75–86. MIT Press, Cambridge, MA.

Cantara, W.A., P.F. Crain, J. Rozenski, J.A.

McCloskey, K.A. Harris, X. Zhang, F.A.P. Vendeix, D. Fabris, and P.F. Agris. 2011. The RNA modification database, RNAMDB: 2011 update. *Nucleic Acids Research* 39(suppl 1): D195-D201.

Capelson, M., and V.G. Corces. 2004. Boundary elements and nuclear organization. *Biology of the Cell* 96(8):617–29.

Carey, N. 2012. *The Epigenetics Revolution: How Modern Biology Is Rewriting Our Understanding of Genetics, Disease and Inheritance*. Icon Books Ltd., London, UK.

Caydasi, A.K., B. Ibrahim, and G. Pereira. 2010. Monitoring spindle orientation: spindle position checkpoint in charge. *Cell Division* 5:28.

Cech, T.R. 2002. Ribozymes, the first 20 years. *Biochemical Society Transactions* 30(6):1162–1166.

Cessac, B., H. Paugam-Moisy, and T. Viéville. 2010. Overview of facts and issues about neural coding by spikes. *Journal of Physiology* – Paris 104:5–18.

Chambon, P. 1995. The molecular and genetic dissection of the retinoid signaling pathway. *Recent Progress in Hormone Research* 50:317–332.

Chen, Y., and J. Jiang. 2013. Decoding the phosphorylation code in Hedgehog signal transduction. *Cell Research* 23(2): 186–200.

Chen, Y.F., N. Etheridge, and G.E. Schaller. 2005. Ethylene signal transduction. *Annals of Botany* 95:901–915.

Cheng et al., 2012. Understanding transcriptional regulation by integrative analysis of transcription factor binding data. *Genome Research* 22:1658–1667.

Chu, H.Y., et al. (2011). From hormones to secondary metabolism: the emergence of metabolic gene clusters in plants, *Plant Journal* 66(1):66–79.

Chung, W.Y., S. Wadhawan, R. Szklarczyk, S.K. Pond, and A. Nukrutenko. 2007. A first look at ARFome: dual-coding genes in mammalian genomes. *PLoS Computational Biology* 3(5): e91.

Ciapponi, L., and G. Cenci. 2008. Telomere capping and cellular checkpoints: clues from fruit flies. *Cytogenetic and Genome Research* 122(3–4):365–373.

Cieśla, J., T. Frączyk, and W. Rode. 2011. Phosphorylation of basic amino acid residues in proteins: important but easily missed. *Acta Biochimica Polonica* 58(2): 137–147.

Claverys, J.P., M. Prudhomme, and B. Martin. 2006. Induction of competence regulons as a general response to stress in gram-positive bacteria. *Annual Review of Microbiology* 60:451–475.

Cosgrove, M.S., and C. Wolberger. 2005. How does the histone code work? *Biochemistry and Cell Biology* 83:468–476.

Crick, F. 1970. Central dogma of molecular biology. *Nature*, 227:561–563.

Davidson, E.H. 2006. *The Regulatory Genome: Gene Regulatory Networks in Development and Evolution*. Academic Press, London, UK.

Davidson, E.H., and D.H. Erwin. 2006. Gene regulatory networks and the evolution of animal body plans. *Science* 311(5762):796–800.

Dawkins, Richard. 1996. *The Blind Watchmaker*. W. W. Norton & Company, Inc., London, UK.

de Lange, T. 2010. How shelterin solves the telomere end-protection problem. *Cold Spring Harbor Symposia on Quantitative Biology* 75:167–77.

Dill, K.A., S.B. Ozkan, M.S. Shell, and T.R. Weikl. 2008. The protein folding problem. *Annual Review of Biophysics* 37:289–316.

Dobrindt, U., B. Hochhut, U. Hentschel, and J. Hacker. 2004. Genomic islands in pathogenic and environmental microorganisms. *Nature Reviews Microbiology* 2:414–424.

Dufour, Y.S., P.J. Kiley, and T.J. Donohue. 2010. Reconstruction of the core and extended regulons of global transcription factors. *PLoS Genetics* 6(7): e1001027.

Dumesic, P.A., P. Natarajan, C. Chen, I.A. Drinnenberg, B.J. Schiller, J. Thompson, J.J. Moresco, J.R. Yates III, D.P. Bartel, and H.D. Madhani. 2013. Stalled spliceosomes signal for RNAi-mediated genome defense. *Cell* 152(5):957–68.

Emanuelsson, O. 2002. Predicting protein subcellular localisation from amino acid sequence information. *Briefings in Bioinformatics* 3(4):361–376.

Engelberg-Kulka, H., I. Yelin, and I. Kolodkin-Gal. 2009. Activation of a built-in bacterial programmed cell death system as a novel mechanism of action of some antibiotics. *Communicative & Integrative Biology* 2(3):211–212.

Fang, S.C., C. de los Reyes, and J.G. Umen. 2006. Cell size checkpoint control by the retinoblastoma tumor suppressor pathway. *PLoS Genetics* 2(10): e167.

Fell, D. 1997. *Understanding the Control of Metabolism*. Portland Press Ltd., London, UK.

Ficz, G. 2015. New insights into mechanisms that regulate DN methylation patterning. *The Journal of Experimental Biology* 218:14–20.

Flames, N., R. Pla, D.M. Gelman, J.L.R. Rubenstein, L. Puelles, and O. Marin. 2007. Delineation of multiple subpallial progenitor domains by the combinatorial expression of transcriptional codes. *The Journal of Neuroscience* 27(36): 9682–9695.

Forsberg, E.C., and S. Smith-Berdan. 2009. Parsing the niche code: the molecular mechanisms governing hematopoietic stem cell adhesion and differentiation. *Haematologica* 94(11):1477–1481.

Fuqua, C., M.R. Parsek, and E.P. Greenberg. 2001. Regulation of gene expression by cell-to-cell communication: Acyl-homoserine lactone quorum sensing. *Annual Review of Genetics* 35:439–468.

Gabdank, I., et al. 2010. FineStr: a web server for single-base-resolution nucleosome positioning. *Bioinformatics* 26:845–846.

Gabius, H.-J., S. André, J. Jiménez-Barbero, A. Romero, and D. Solís. 2011. From lectin structure to functional glycomics: principles of the sugar code. *Trends in Biochemical Sciences* 36(6):298–313.

Gatlin, L.L. 1972. *Information Theory and the Living System*. Columbia University Press, New York, NY.

Gazzaniga, M.S., R.B. Ivry, and G.R. Mangun. 2009. *Cognitive Neuroscience: The Biology of the Mind*, 3rd edition. W. W.

Norton & Company, Inc., New York, NY.

Gen New Highlights. 2015. Bacteria use CRISPR to recognize themselves. http://www.genengnews.com/gen-news-highlights/bacteria-use-crispr-to-recognize-themselves/81251146/

Gibbs, D.J., J. Bacardit, A. Bachmair, and M.J. Holdsworth. 2014. The eukaryotic N-end rule pathway: conserved mechanisms and diverse functions. *Trends in Cell Biology*, 24(10):603–611.

Gilbert, S.F. 2003. The reactive genome. In Müller, G.B., and S.A. Newmann (editors), *Origination of Organismal Form: Beyond the Gene in Development and Evolutionary Biology*, pp. 87–102. MIT Press, Cambridge, MA.

Gogarten, J.P., A.G. Senejani, O. Zhaxybayeva, L. Olendzenski, and E. Hilario. 2002. Inteins: structure, function, and evolution. *Annual Review of Microbiology* 56:263–287.

Gurdon, J.B., J.A. Byrne, S. Simonsson. 2003. Nuclear reprogramming and stem cell creation. *Proceedings of the National Academy of Sciences USA* 100 (Suppl 1):11819–11822.

Hall, R.M., and C.M. Collis. 1995. Mobile gene cassettes and integrons: capture and spread of genes by site-specific recombination. *Molecular Microbiology* 15(4):593–600.

Harrow, J., A. Nagy, A. Reymond, T. Alioto, L. Patthy, S.E Antonarakis, and R. Guigó. 2009. Identifying protein-coding genes in genomic sequences. *Genome Biology* 10:201.

Hegde, R.S., and H.D. Bernstein. 2006. The surprising complexity of signal sequences. *Trends in Biochemical Sciences* 31(10):563–71.

Hofstadter, D.R. 1980. *Gödel, Escher, Bach: An Eternal Golden Braid*. Random House/Vintage Books, New York, NY.

Holoch, P.A., and T.S. Griffith. 2009. TNF-related apoptosis-inducing ligand (TRAIL): a new path to anti-cancer therapies. *European Journal of Pharmacology* 625(1–3):63–72.

Hou, Y.M., and P. Schimmel. 1988. A simple structural feature is a major determinant of the identity of a transfer RNA. *Nature* 333:140–145.

Huen, M.S., and J. Chen. 2010. Assembly of checkpoint and repair machineries at DNA damage sites. *Trends in Biochemical Sciences* 35(2):101–108.

Hughes, T. 2008. Cracking the second genetic code. *FASEB Journal* 22:262.2.

Hurst, L.D., C. Pál, and M.J. Lercher. 2004. The evolutionary dynamics of eukaryotic gene order. *Nature Reviews Genetics* 5:299–310.

Ibarra, B., et al., 2009. Proofreading dynamics of a processive DNA polymerase. *EMBO Journal* 28(18):2794–2802.

Ikegami, K., J. Ohgane, S. Tanaka, S. Yagi, and K. Shiota. 2009. Interplay between DNA methylation, histone modification and chromatin remodeling in stem cells and during development. *International Journal of Developmental Biology* 53:203–214.

Ishihama, A. 2000. Functional modulation of Escherichia coli RNA polymerase. *Annual Review of Microbiology* 54:499–518.

Ishikawa, K., H. Ishii, and T. Saito. 2006. DNA damage-dependent cell cycle checkpoints and genomic stability. *DNA and Cell Biology* 25:406–411.

Jády, B.E., P. Richard, E. Bertrand, and T. Kiss. 2006. Cell cycle-dependent recruitment of telomerase RNA and Cajal bodies to human telomeres. *Molecular Biology of the Cell* 17 (2):944–54.

Jaenisch, R., and A. Bird. 2003. Epigenetic regulation of gene expression: how the genome integrates intrinsic and environmental signals. *Nature Genetics* 33 (Suppl):245–254.

Janke, C. 2014. The tubulin code: molecular components, readout mechanisms, and functions. *Journal of Cell Biology* 206(4):461–472.

Jenuwein, T., and C.D. Allis. 2001. Translating the histone code. *Science* 293:1074–1080.

Jessell, T.M. 2000. Neuronal specification in the spinal cord: inductive signals and transcriptional codes. *Nature Genetics* 1:20–29.

Jolma, A., Y. Yin, K.R. Nitta, K. Dave, A. Popov, M. Taipale, M. Enge, T. Kivioja, E. Morgunova, and J. Taipale. 2015. DNA-dependent formation of transcription factor pairs alters their binding specificity. *Nature* 527:384–388.

Jones, P.A. 2012. Functions of DNA methylation: islands, start sites, gene bodies and beyond. *Nature Reviews Genetics* 13(7):484–92.

Juven-Gershon, T., and J.T. Kadonaga. 2010. Regulation of gene expression via the core promoter and the basal transcriptional machinery. *Developmental Biology* 339(2):225–229.

Kadyrova, L.Y., et al. 2013. A reversible histone h3 acetylation cooperates with mismatch repair and replicative polymerases in maintaining genome stability. *PLoS Genetics* 9:e1003899.

Keverne, E.B., D.W. Pfaff, and I. Tabansky. 2015. Epigenetic changes in the developing brain: effects on behavior. *Proceedings of the National Academy of Sciences* 112(22):6789–6795.

Kirschner, M.W., and J.C. Gerhart. 2005. *The Plausibility of Life: Resolving Darwin's Dilemma*. Yale University Press, New Haven, CT.

Klipp, E., W. Liebermeister, C. Wierling, A. Kowald, H. Lehrach, and R. Herwig. 2009. *Systems Biology*. Wiley-VCH Verlag GmbH & Co. KgaA, Weinheim, Germany.

Kolovos, P., T.A. Knoch, F.G. Grosveld, P.R. Cook, and A. Papantonis. 2012. Enhancers and silencers: an integrated and simple model for their function. *Epigenetics & Chromatin* 5:1–8.

Komano, T. 1999. Shufflons: multiple inversion systems and integrons. *Annual Review of Genetics* 3:171–191.

Koonin, E.V., K.S. Makarova, and L. Aravind. 2001. Horizontal gene transfer in prokayrotes: quantification and classification. *Annual Review of Microbiology* 55:709–742.

Kosko, B., and S. Isaka. 1993. Fuzzy logic. *Scientific American* July:76–81.

Krysko, D.V., et al. 2005. Gap junctions and the propagation of cell survival and cell death signals. *Apoptosis* 10:459–469.

Kundaje, A. et al. 2015. Integrative analysis of 111 reference human epigenomes. *Nature* 518:317–330.

Kunkel, T.A., and K. Bebenek. 2000. DNA replication fidelity. *Annual Review of Biochemistry* 69:497–529.

Kunkel, T.A., and D.A. Erie. 2005. DNA mismatch repair. *Annual Review of Biochemistry* 74:681–710.

Lazazzera, B.A. 2001. The intracellular function of extracellular signaling peptides. *Peptides* 22:1519–1527.

Lee, H.J., T.A. Hore, and W. Reik. 2014. Reprogramming the methylome: erasing memory and creating diversity *Cell Stem Cell* 14:710–719.

London, M., and M. Hausser. 2005. Dendritic computation. *Annual Review of Neuroscience* 28:503–532.

Markram, H., et al. 2015. Reconstruction and simulation of neocortical microcircuitry. *Cell* 163:456–492.

Marquardt, T., and S.L. Pfaff. 2001. Cracking the transcriptional code for cell specification in the neural tube. *Cell* 106:651–654.

Mattick, J.S., and I.V. Makunin. 2006. Non-coding RNA. *Human Molecular Genetics* 15(1):R17-R29.

Michalak, P. 2008. Coexpression, coregulation, and cofunctionality of neighboring genes in eukaryotic genomes. *Genomics* 91(3):243–248.

Miller, K.M., et al. 2010. Human HDAC1 and HDAC2 function in the DNA-damage response to promote DNA nonhomologous end-joining. *Nature Structural and Molecular Biology* 17:1144–1151.

Miwa, Y., A. Nakata, A. Ogiwara, M. Yamamoto, and Y. Fujita. 2000. Evaluation and characterization of catabolite-responsive elements (cre) of Bacillus subtilis. *Nucleic Acids Research* 28:1206–1210.

Modrich, P., and R. Lahue. 1996. Mismatch repair in replication fidelity, genetic recombination, and cancer biology. *Annual Review of Biochemistry* 65:101–133.

Moller-Krull, M., et al. 2008. Beyond DNA: RNA editing and steps toward Alu exonization in primates. *Journal of Molecular Biology* 382(3):601–599.

Montañez, G., R.J. Marks II, J. Fernandez, and J.C. Sanford. 2013. Multiple overlapping genetic codes profoundly reduce the probability of beneficial mutations. In Marks, R.J. II, M.J. Behe, W.A. Dembski, B.L. Gordon, and J.C. Sanford (editors), *Biological Information: New Perspectives*, pp. 139–167. World Scientific, Tuck Link, Singapore.

Müller, G.B., and S.A. Newmann (editors). 2003. *Origination of Organismal Form: Beyond the Gene in Development and Evolutionary Biology*. MIT Press, Cambridge, MA.

Murai, K.K., and E.B. Pasquale. 2004. Eph receptors, ephrins, and synaptic function. *Neuroscientist* 10:304–314.

Murphy, P.V., S. André, and H.-J. Gabius. 2013. The third dimension of reading the sugar code by lectins: design of glycoclusters with cyclic scaffolds as tools with the aim to define correlations between spatial presentation and activity. *Molecules* 18:4026–4053.

Musacchio, A. 2011. Spindle assembly checkpoint: the third decade. *Philosophical Transactions of the Royal Society of London B, Biological Sciences* 366(1584): 3595–604.

Neel, N.F., E. Schutyser, J. Sai, G.H. Fan, and A. Richmond. 2005. Chemokine receptor internalization and intracellular trafficking. *Cytokine and Growth Factor Reviews* 16:637–658.

Nezi, L., and A. Musacchio. 2009. Sister chromatid tension and the spindle assembly checkpoint. *Current Opinion in Cell Biology* 21:785–795.

Nguyen, V.C., et al. 2010. Replication stress checkpoint signaling controls tRNA gene transcription. *Nature Structural and Molecular Biology* 17(8):976–81.

Nicolelis, M.A.L., and S. Ribeiro. 2006. Seeking the neural code. *Scientific American* 295:70–77.

Nishikura, K. 2010. Functions and regulation of RNA editing by ADAR deaminases. *Annual Review of Biochemistry* 79:321–349.

Noble, D. 2015. Evolution beyond neo-Darwinism: a new conceptual framework. *Journal of Experimental Biology* 218:7–13.

Ochman, H., J.G. Lawrence, and E.A. Groisman. 2000. Lateral gene transfer and the nature of bacterial innovation. *Nature* 405:299–304.

Osbourn, A.E., and B. Field. 2009. Operons. *Cellular and Molecular Life Sciences* 66(23):3755–3775.

Oyama, S. 2002. *The Ontogeny of Information: Developmental Systems and Evolution*. 2nd edition. Duke University Press, Durham, NC.

Page, A.M., and P. Hieter. 1999. The anaphase-promoting complex: new subunits and regulators. *Annual Review of Biochemistry* 68:583–609.

Paszkowski, J., and S.A. Whitham. 2001. Gene silencing and DNA methylation processes. *Current Opinion in Plant Biology* 4:123–129.

Pauli, A., J.L. Rinn, and A.F. Schier. 2011. Non-coding RNAs as regulators of embryogenesis. *Nature Reviews* 12:136–149.

Pelkmans, L. 2005. Viruses as probes for systems analysis of cellular signaling, cytoskeleton reorganization and endocytosis. *Current Opinion in Microbiology* 8:331–337.

Pollard, T.D., and G.G. Borisy. 2003. Cellular motility driven by assembly and disassembly of actin filaments. *Cell* 112:453–465.

Pool, M.R. 2005. Signal recognition particles in chloroplasts, bacteria, yeast and mammals (review). *Molecular Membrane Biology* 22:3–15.

Putnam, C.D., E.J. Jaehnig, and R.D. Kolodner. 2009. Perspectives on the DNA damage and replication checkpoint responses in Saccharomyces cerevisiaem. *DNA Repair* 8(9):974–82.

Ran, F.A., et al. 2015. In vivo genome editing using Staphylococcus aureus Cas9. *Nature* 520(7546):186–191.

Readies, C., and M. Takeichi. 1996. Cadherine in the developing central nervous system: an adhesive code for segmental and functional subdivisions. *Developmental Biology* 180:413–423.

Rino, J., and M. Carmo-Fonseca. 2009. The spliceosome: a self-organized macromolecular machine in the nucleus? *Trends in Cell Biology* 19(8):375–84.

Sabelli, P.A., et al. 2013. Control of cell proliferation, endoreduplication, cell size, and cell death by the retinoblastoma-related pathway in maize endosperm. *Proceedings of the National Academy of Sciences USA* 110:E1827–1836.

Salih, F., et al. 2008. Epigenetic nucleosomes: Alu sequences and CG as nucleosome positioning element. *Journal of Biomolecular Structure and Dynamics* 26:9–16.

Schmidt, C.K., and S.P. Jackson. 2013. On your mark, get SET(D2), go! H3K36me3 primes DNA mismatch repair. *Cell* 153:513–515.

Schneider, R., and R. Grosschedl. 2007. Dynamics and interplay of nuclear architecture, genome organization, and gene expression. *Genes and Development* 21:3027– 3043.

Schübeler, D. 2015. Function and information content of DNA methylation. *Nature* 517:321–326.

Scruton, R. 1996. *A Short History of Modern Philosophy*. Routledge, Abingdon-on-Thames, UK.

Searle, J.S., et al. 2011. Proteins in the nutrient-sensing and DNA damage checkpoint pathways cooperate to restrain mitotic progression following DNA damage. *PLoS Genetics* 7(7):e1002176.

Segal, E., et al. 2006. A genomic code for nucleosome positioning. *Nature* 442:772–778.

Segurado, M., and J.A. Tercero. 2009. The S-phase checkpoint: targeting the replication fork. *Biology of the Cell* 101:617–627.

Serganov, A., and D.J. Patel. 2007. Ribozymes, riboswitches and beyond: regulation of gene expression without proteins. *Nature Reviews Genetics* 8:776–790.

Shapiro, J.A. 2006. Genome informatics: the role of DNA in cellular computations. *Biological Theory* 1(3):288–301.

Shapiro, J.A. 2011. Cognitive aspects of genome function. Symposium on Neurobiological Correlates of Interpersonal Relations, Freiburg, Germany, October 15.

Shapiro, J.A. 2014. Physiology of the read-write genome. *Journal of Physiology* 592.11:2319–2341.

Shapiro, L., and D.R. Colman. 1999. The diversity of cadherins and implications for a synaptic adhesive code in the CNS. *Neuron* 23:427–430.

Shapiro, J.A., and R. von Sternberg. 2005. Why repetitive DNA is essential to genome function. *Biological Reviews* 80:1–24.

Sidiropoulou, K, E.K. Pissadaki, and P. Poirazi. 2006. Inside the brain of a neuron. *EMBO Reports* 7(9):886–892.

Signal Peptide Website: An Information Platform for Signal Sequences and Signal Peptides. http://www.signalpeptide.de/

Sonea, S., and L.G. Mathieu. 2001. Evolution of the genomic systems of prokaryotes and its momentous consequences. *International Microbiology* 4:67–71.

Song, J. 2007. EMT or apoptosis: a decision for TGF-beta. *Cell Research* 17(4): 289–290.

Spencer, P.S., E. Siller, J.F. Anderson, and J.M. Barral. 2012. Silent substitutions predictably alter translation elongation rates and protein folding efficiencies. *Journal of Molecular Biology* 422:328–335.

Stangroom, J., and J. Garvey. 2005. *The Great Philosophers: From Socrates to Foucault*. Arcturus Publishing Limited, Longdon, UK.

Stanton, T.B. 2007. Prophage-like gene transfer agents—novel mechanisms of gene exchange for Methanococcus, Desulfovibrio, Brachyspira, and Rhodobacter species. *Anaerobe* 13:43–49.

Storz, G., Altuvia, S., and K.M. Wassarman. 2005. An abundance of RNA regulators. *Annual Review of Biochemistry* 74:199–217.

Strahl, B.D., and C.D. Allis. 2000. The language of covalent histone modifications. *Nature* 403:41–45.

Striedter, G.F. 2003. Epigenesis and evolution of brains: from embryonic divisions to functional systems. In Müller, G.B., and S.A. Newmann (editors), *Origination of Organismal Form: Beyond the Gene in Development and Evolutionary Biology*, pp. 287–304. MIT Press, Cambridge, MA.

Stuart, L.M., R.A. Ezekowitz. 2005. Phagocytosis: elegant complexity. *Immunity* 22:539–550.

Sugiyama, T., et al. 2005. RNA-dependent RNA polymerase is an essential component of a self-enforcing loop coupling heterochromatin assembly to siRNA production. *Proceedings of the National Academy of Sciences USA* 102(1): 152–157.

Sullivan, B.A., M.D. Blower, and G.H. Karpen. 2001. Determining centromere identity: cyclical stories and forking paths. *Nature Reviews Genetics* 2:584–596.

Taft, R.J., K.C. Pang, T.C. Mercer, M. Dinger, and J.S. Mattick. 2010. Non-coding RNAs: regulators of disease. *Journal of Pathology* 220:126–139.

Takada, Y., X. Ye, and S. Simon. 2007. The integrins. *Genome Biology* 8(5):215.

Tam, C.K.P., J. Hackett, and C. Morris. 2005. Rate of inversion of the Salmonella enterica shufflon regulates expression of invertible DNA. *Infection and Immunity* 73(9):5568–5577.

Tejedor, J.R., and J. Valcárcel. 2010. Breaking the second genetic code. *Nature* 465(6):45–46.

Tentner, A.R., et al. 2012. Combined experimental and computational analysis of DNA damage signaling reveals context-dependent roles for Erk in apoptosis and G1/S arrest after genotoxic stress. *Molecular Systems Biology* 8:568.

Thanbichler, M., and L. Shapiro. 2008. Getting organized–how bacterial cells move proteins and DNA. *Nature Reviews Microbiology* 6:28–40.

Thomas, C.M., and K.M. Nielsen. 2005. Mechanisms of, and barriers to, horizontal gene transfer between bacteria. *Nature Review Microbiology* 3:711–721.

Togneri, R., and C.J.S. deSilva. 2003. *Fundamentals of Information Theory and*

Coding Design. Chapman & Hall/CRC, Boca Raton, FL.

Tomkins, M.G. 1975. The metabolic code. *Science* 189:760–763.

Trifonov, E.N. 1980. Sequence-dependent deformational anisotropy of chromatin DNA. *Nucleic Acids Research* 8:4041–4053.

Trifonov, E.N. 1981. Structure of DNA in chromatin. In Schweiger, H. (editor), *International Cell Biology 1980–1981*, pp. 128–138. Springer, Berlin, Germany.

Trifonov, E.N. 2011. Thirty years of multiple sequence codes. *Genomics Proteomics Bioinformatics* 9(1–2):1–6.

Trotta, C.R., F. Miao, E.A. Arn, S.W. Stevens, C.K. Ho, R. Rauhut, and J.N. Abelson. 1997. The yeast tRNA splicing endonuclease: a tetrameric enzyme with two active site subunits homologous to the archaeal tRNA endonucleases. *Cell* 89(6):849–858.

Truman, R. 2012. An evaluation of codes more compact than the natural genetic code. *Journal of Creation* 26(2):88–99.

Truman, R. 2012a. Information theory—part 1—overview of key ideas. *Journal of Creation* 26(3):101–106. http://creation.com/cis-1.

Truman, R. 2012b. Information theory—part 2: weaknesses in current conceptual frameworks. *Journal of Creation* 26(3):107–114. http://creation.com/cis-2.

Truman, R. 2012c. Information theory—part 3: introduction to coded information systems. *Journal of Creation* 26(3):115–119. http://creation.com/cis-3.

Truman, R. 2013. Information theory—part 4: fundamental theorems of coded information systems theory. *Journal of Creation* 27(1):71–77. http://creation.com/cis-4.

Truman, R. 2015. Nylon-eating bacteria—part 4: interpretation according to coded information system theory. *Journal of Creation* 29(3):46–52.

van de Lagemaat, L.N., L. Gagnier, P. Medstrand, and D.L. Mager. 2005. Genomic deletions and precise removal of transposable elements mediated by short identical DNA segments in primates. *Genome Research* 15(9):1243–1249.

Varshavsky A. 2011. The N-end rule pathway and regulation by proteolysis. *Protein Science* 8:1298–345.

Verdel, A., et al. 2009. Common themes in siRNA-mediated epigenetic silencing pathways. *International Journal of Developmental Biology* 53(2–3):245–257.

Verhey, K.J., and J. Gaertig. 2007. The tubulin code. *Cell Cycle* 6:(17):2152–2160.

Wagner, S., et al. 2015. The EF-hand Ca2 binding protein MICU choreographs mitochondrial Ca2 dynamics in arabidopsis. *The Plant Cell* 27(11):3190–3212.

Walhout, M., M. Vidal and J. Dekker (editors). 2013. *Handbook of Systems Biology: Concepts and Insights*. Elsevier, London, UK.

Walsh, C.M., and A.L. Edinger. 2010. The complex interplay between autophagy, apoptosis, and necrotic signals promotes T-cell homeostasis. *Immunological Reviews* 236:95–109.

Wellik, D.M. 2007. Hox patterning of the vertebrate axial skeleton. *Development Dynamcs* 236:2454–2463.

Widelitz, R. 2005. Wnt signaling through canonical and non-canonical pathways: recent progress. *Growth Factors* 23:111–116.

Wikipedia. n.d. Von Neumann architecture. https://en.wikipedia.org/wiki/Von_Neumann_architecture.

Williams, T.L., D.L. Levy, S. Maki-Yonekura, K. Yonekura, and E.H. Blackburn. 2010. Characterization of the yeast telomere nucleoprotein core. Rap1 binds independently to each recognition site. *The Journal of Biological Chemistry* 285:35814–35824.

Willmer, P. 2003. Convergence and homplasmy in the evolution of organizmal form. In Müller, G.B., and S.A. Newmann (editors), *Origination of Organismal Form: Beyond the Gene in Development and Evolutionary Biology*, pp. 33–50. MIT Press, Cambridge, MA.

Woodward, T.E., and J.P. Gills. 2012. *The Mysterious Epigenome: What Lies Beyond DNA*. Kregel Publications, Grand Rapids, MI.

Wulfing C., I. Tskvitaria-Fuller, N. Burroughs, M.D. Sjaastad, J. Klem, J.D. Schatzle. 2002. Interface accumulation of receptor/ligand couples in lymphocyte activation: Methods, mechanisms, and significance. *Immunological Reviews* 189:64–83.

Yamada, S., and W.J. Nelson. 2007. Synapses: Sites of cell recognition, adhesion, and functional specification. *Annual Review of Biochemistry* 76:267–294.

Young, E. 2001. Packaging proteins may be second genetic code. *New Scientist*, August 9. https://www.newscientist.com/article/dn1140-packaging-proteins-may-be-second-genetic-code/

Yuh, C.H., H. Bolouri, and E.H. Davidson. 1998. Genomic cis-regulatory logic: experimental and computational analysis of a sea urchin gene. *Science* 279:1896–1902.

Zeiler, M., W.L. Straube, E. Lundberg, M. Uhlen, and M. Mann. 2012. A protein epitope signature tag (PrEST) library allows SILAC-based absolute quantification and multiplexed determination of protein copy numbers in cell lines. *Molecular & Cellular Proteomics* 11(3): O111.009613.

Zetsche, B., et al. 2015. Cpf1 is a single RNA-guided endonuclease of a class 2 CRISPR-cas system. *Cell* 163:759–771.

Zhou, T., N. Shen, L. Yang, N. Abe, J. Horton, R.S. Mann, H.J. Bussemaker, R. Gordân, and R. Rohs. 2015. Quantitative modeling of transcription factor binding specificities using DNA shape. *Proceedings of the National Academy of Sciences USA* 112(15):4654–4659.