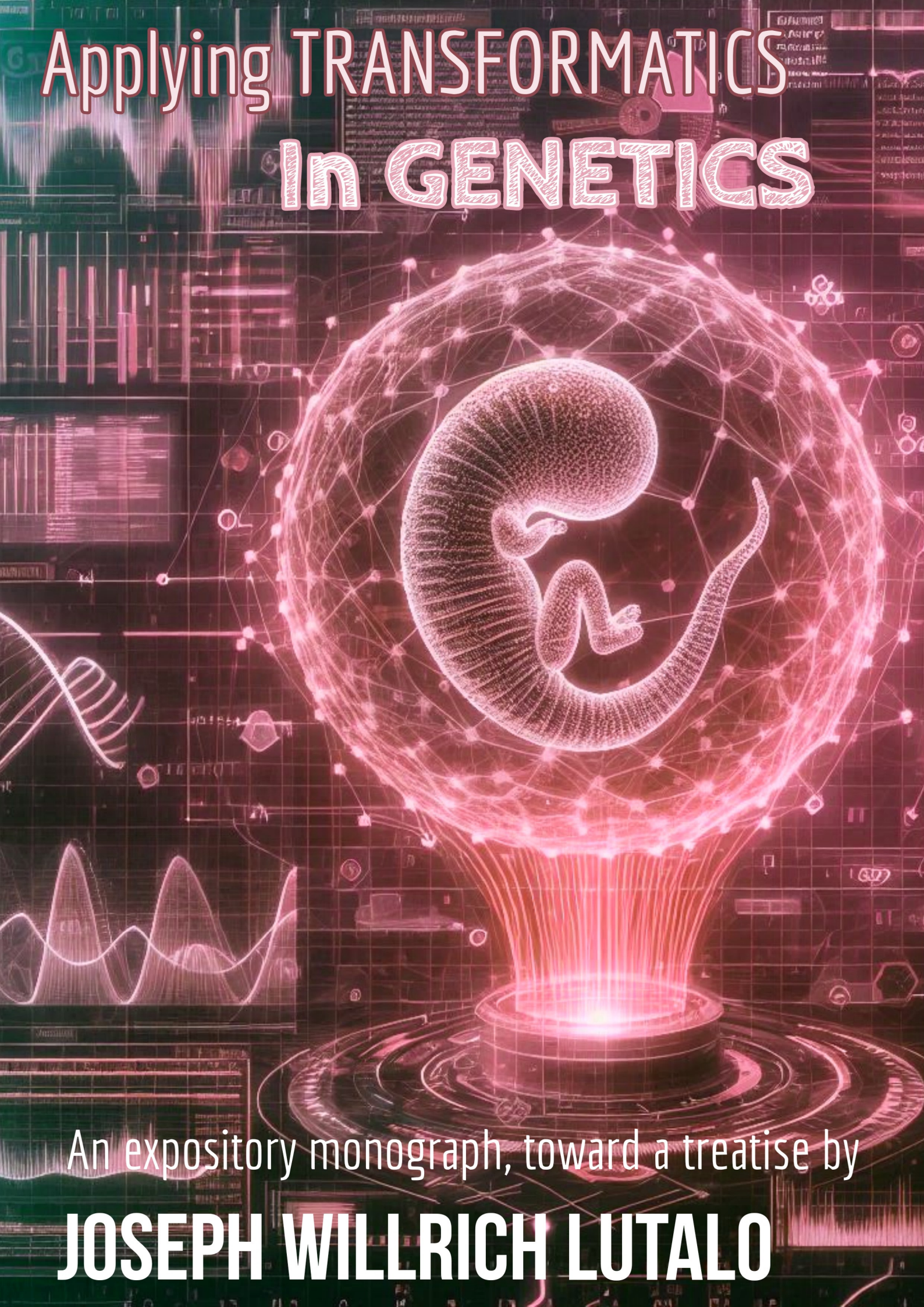
Applying TRANSFORMATICS In GENETICS



An expository monograph, toward a treatise by

JOSEPH WILLRICH LUTALO

Applying **TRANSFORMATICS** in GENETICS

Willrich J. Lutalo[1]
joewillrich@gmail.com, jwl@nuchwezi.com

August 18, 2025

[1]Currently a volunteering & Independent Principal Investigator at Nuchwezi Research — https://nuchwezi.com

Contents

Preface **ix**

1 Introduction **3**
1.0.1 How to Navigate the Monograph . . . 6

2 Sequence Symbol Sets and Sequence Transforms Applied to Genetic Code **9**

3 Sequence Abstraction using Symbol Sets: From Flat-Structure Sequences to Higher-Level Sequences **13**

4 Sequence Analysis Using Symbol Sets **15**

5 Sequence Analysis Using Complement Sets **19**
5.1 Significance of Sequence Complements, $\neg\Theta$, and Complement Symbol Sets, $\neg\psi_\beta$ . . . 21
5.2 Potential Applications of Sequence Complements or Inverses . . . 23

6 Sequence Analysis Using The Anagram Distance and the Modal Sequence Statistic **27**
6.1 Six Sequences and Four Scenarios . . . 27
6.2 A First Analysis . . . 30
6.3 SCENARIO A — the anagram distance measure . . . 33
6.4 SCENARIO B — the sequence characteric [statistic] . . . 36
6.5 SCENARIO C . . . 39
6.6 SCENARIO D — the population characteric measure . . . 40
6.6.1 Measuring Proximity to a Set of Sequences, Ω^n . . . 40
6.6.2 Measuring Proximity between a Q_k and Members of a Sequence Population, Ω^n . . . 43
6.7 Concerning Optimal Symbolic Storage of na-Sequences in Databases - Computing the Gödel Number for a Sequence Θ_n based on its Characteristic $\hbar(\overset{>}{\Theta}_n)$ . . . 45

7 The Mathematics of Genome Sequencing: Sequence Alignment, The Modal Sequence, The Sequencing Machine and the Identifier Genome Sequence Law **49**
7.1 PROBLEM G1: Computing MCS: Maximum Common Subsequence: $\Theta_1 \lozenge \Theta_2$ . . . 50
7.2 PROBLEM G2: Computing Overlap Resultant Sequences: $\Theta_1 * \Theta_2$ . . . 52
7.3 PROBLEM G3: Computing the Genome Sequence (GS): $(\Theta_1 \lozenge \Theta_2)^*$ . . . 52
7.4 The Genome Sequencer . . . 54
7.5 The Identity Genome Sequence (IGS) Law . . . 54

8 Random Sequence Generators (RSGs) **57**
8.1 The First Lu-Shuffle Algorithm (FLSA) . . . 57
8.1.1 The FLSA Algorithm Using Mathematics . . . 57
8.1.2 Examples of Applying FLSA . . . 58
8.1.3 The Source Code for FLSA . . . 60
8.2 The Second Lu-Shuffle Algorithm (SLSA) . . . 62
8.2.1 The SLSA Algorithm Using Mathematics . . . 62
8.2.2 Examples of Applying SLSA . . . 63
8.2.3 The Source Code for SLSA . . . 65
8.3 Generating Realistic Random na-Sequences . . . 66
8.3.1 How to Generate a Random Nucleotide & The Random Symbol Generator . . . 66
8.3.2 The USYMBOLSET Function . . . 68
8.3.3 The PROTRACT Function . . . 68
8.3.4 The Random Number Generator, RNG Function . . . 69

8.3.5 The SHUFFLE Function . . . 71
8.3.6 The SELECT Function . . . 72
8.3.7 The RSG Function . . . 73

9 Gene Expression in Living Organisms Leveraging Genetic Code (DNA $\rightarrow mRNA \rightarrow Protein \rightarrow$ Organism) 77
9.1 Protein Manufacturing Algorithm . . . 82
9.2 Gene Expression in Living Things and for Bio-Automata via Ribosomes . . . 85

10 Transformatics and Lu-Genome Expression System in Bio-Automata 89
10.1 The Ozin Genetic Code Cipher . . . 90
10.1.1 Ω-to-OZIN Genetic Code Transcription Rules . . . 93
10.1.2 Examples of Ω-Genetic Code Transcription . . . 94
10.2 Genetic Code Sequences Processed as Modal Sequence Statistics . . . 95
10.2.1 The Platonic Form Cipher and Position-Sensitive Genetic Code Translation . . . 96
10.2.2 ψ_Ω-to-ψ_{pf}: PLATONIC Form Genetic Code Translation Rules . . . 97
10.2.3 Examples of Complete Lu-Genome Expressions . . . 101
10.3 The Extended Lu-Genome Expression System (x-LGES) . . . 106
10.3.1 ψ_Ω-to-(ψ_{oz}-ψ_{pf}-ψ_{oz}): The x-LGES Algorithm . . . 107
10.4 Actual DNA Sequences Expressed in the Extended Lu-Genome Expression System . . . 109
10.4.1 Converting Between Nucleic Acid Code and Numeric Codes . . . 109
10.4.2 Revisiting Numeric Sequences . . . 112
10.4.3 x-LGES applied to named DNA Sequences . . . 113

11 Conclusion 115

Appendix A Origins of the OZIN Cipher 121

Appendix B Computing Platonic-Form Encoding Keys (PFEKs), $\boxed{\psi_\Omega}_m$ for a Particular na-Sequence Θ or m 127
B.0.1 Example of Resolving PFA Components, $\boxed{\omega_i^*}$ . . . 129

Appendix C Alternative[ORIGINAL] Ω-to-PLATONIC Form Genetic Code Expression Algorithm 133
C.0.1 Examples of Complete Lu-Genome Expressions . . . 137

List of Figures

1.1 In the course of writing this work, an experiment to grow pineapples from the pineapple fruit tops of pineapples normally bought from nearby marketplace (**Garuga/Bugabo-Bukaaya**) yielded results! Here we see the first successful harvest of a pineapple from our home garden (at **Nuchwezi Research**). But also, this case inspired part of the discussion in **Chapter 10** about how in real genome expression systems, the organism expressed from genetic code somewhat still includes the genetic code in the manifested organism as some kind of affix — a prefix (in leaves for example) or suffix (roots), from which the original organism (or the same exact genome) might be later reproduced! 3
1.2 A basic diagram of the chemical structure of DNA base-pairs from [1] 5

6.1 Some proposal to use Gödel's idea of abstracting formulae such as formal symbolic sequences using distinct natural numbers. 45

8.1 A Tabulation Trace of Shuffling the Base-10 n-SSI Sequence with increasing values of k, the **Initial Partitioning Index** for **Algorithm 3**, the **First Lu-Shuffle Algorithm**. 59
8.2 A Tabulation Trace of Shuffling basic base-case scenarios to help highlight how **FLSA** works. . . . 59
8.3 A Tabulation Trace of Shuffling the na-Sequence Θ_{2na} from **Equation 10.27** with **Algorithm 3 (FLSA)**. 60
8.4 A Tabulation Trace of Shuffling the na-Sequence Θ_{palna} from **Equation 10.28** with **Algorithm 3 (FLSA)**. 60
8.5 The First Lu-Shuffle Algorithm implemented using Python 61
8.6 A Tabulation Trace of Shuffling the Base-10 n-SSI Sequence with increasing values of k, using **Algorithm 4 (SLSA)**, the **Second Lu-Shuffle Algorithm**. 63
8.7 A Tabulation Trace of Shuffling basic base-case scenarios to help highlight how **SLSA** works and how it relates to **FLSA**. 64
8.8 A Tabulation Trace of Shuffling the na-Sequence Θ_{2na} from **Equation 10.27** with **Algorithm 4 (SLSA)**. 64
8.9 A Tabulation Trace of Shuffling the na-Sequence Θ_{palna} from **Equation 10.28** with **Algorithm 4 (SLSA)**. 65
8.10 The Second Lu-Shuffle Algorithm implemented using Python 66
8.11 The **USYMBOLSET**(Ω) unspecific symbol set generator implemented using Python 68
8.12 The **PROTRACT**($\psi_y, 2x$) sequence generator-transformer implemented using Python 69
8.13 The **RNG**(ll, ul) random number generator implemented using Python 70
8.14 A text dump from testing our RNG(ll, ul) that only utilizes System Time as entropy source, and uses our First Lu-Shuffle Algorithm to generate True Random Numbers 71
8.15 The **SHUFFLE**(Θ) sequence randomizer implemented using Python 72
8.16 The **SELECT**(n, Θ) sequence sampler function implemented using Python 73
8.17 The **RSG**(n, Θ) random sequence generator function implemented using Python 74
8.18 A Random DNA Sequence of exactly 200 random codons as generated using our **RSG**(n, Θ) random sequence generator algorithm. 75

9.1 An illustration of the gene translation process in a cell via protein-factories known as ribosomes[2] 77
9.2 **Flow-Chart summarizing the Ribosome-based Protein-Synthesis Process in a Living Cell** 83
9.3 **The RIBOSOME State Machine** 84

10.1 The OZIN Cipher[3], a mapping from Decimal Symbols (our base-Ω) to Symbols for Intermediate Genetic Code. 91
10.2 The equivalent OZIN expression of the hypothetical Euler Virus genome sequence. 92
10.3 The equivalent OZIN expression of the HiFinelle genome sequence. 92
10.4 The equivalent OZIN expression of the Ω_3 genome sequence. 93

10.5 The palindromic sequence Ω_{pal} transcribed into base-OZ. 94
10.6 The PLATONIC Cipher[4] a mapping from Decimal Symbols (our base-Ω) to Spatial-Geometric Forms (ψ_{pf}). 97
10.7 An example of a single **amino-acid** or rather, the **PFA** component of a **FGE** equivalent to one **amino-acid** segment expressed using the **LGES** system. This particular expression corresponding to a rendering of the configuration $4_{pf} \otimes 6_{oz} = 4_{pf} + 4 \times 6_{oz}$. 101
10.8 The equivalent LGES complete genome expression of the Ω_3 genome sequence (from **Equation 10.4**) 102
10.9 The equivalent LGES complete genome expression of the Ω_{veuler} (Euler Virus) genome sequence (from **Equation 10.19**) 103
10.10 The **alternative rendering** of the same sequence as Ω_{veuler}. 104
10.11 The **x-LGES form rendering** of the Ω_{veuler}, the **Euler Virus**, now with HEAD, BODY and ROOT segments. 108
10.12 An example of an actual organism from nature — an Earthworm, for which, like the x-LGES rendering of the sequence $\Omega_{HiFinelle}$, results in a genome expression for a creature that almost looks the same at the HEAD and ROOT. Image sourced from Wikimedia Commons[5]. 113

A.1 An excerpt from architectural design notes by the author from early days of his home-based research laboratory. In this note, we also get to see an idea similar to how gene expressions are actually a chain of glyphs that, as in our **LGES** protocols, are depicted as structures chained to each other along a *backbone structure* symbolically depicted as a line (curved lines in this case). Copyrights due to author and Nuchwezi Research Labs. 122
A.2 The OZIN Cipher in use. An excerpt from the Church of Dance Eternal OGF 124
A.3 The OZIN Cipher in use. An excerpt from author's research notes (circa 2019) on developing a cryptographic system leveraging a dial mechanism. Copyrights due to author and Nuchwezi Research Labs. 125

B.1 An example of a single **PFA** component equivalent to the configuration $8_{pf} \otimes 6_{oz}$. 131

C.1 The equivalent LGES complete genome expression of the Ω_3 genome sequence (from **Equation 10.4**) 137
C.2 The equivalent LGES complete genome expression of the Ω_{veuler} (Euler Virus) genome sequence (from **Equation C.12**) 138
C.3 The **alternative rendering** of the same sequence as Ω_{veuler}. 139

List of Tables

5.1 An example of exploring symbol and sequence complements in sequence transformations 20

6.1 A First Analysis of the 5 na-Sequences from **Section 6.1** 33
6.2 A Tabular Analysis of $\Theta 1$ Vs $\Theta 4$ so as to compute their Anagram Distance 35

9.1 Amino-Acid Codes and Names in Θ_4, a DNA-encoded gene 81
9.2 Amino-Acid Codes and Names in Θ_4^*, a mRNA encoded gene 81

B.1 Platonic Form Encoding Keys (PFEK) mapped to corresponding values of $m = \mu(\overset{>}{\Theta})$ using **FLSA** 128
B.2 Platonic Form Encoding Keys (PFEK) mapped to corresponding values of $m = \mu(\overset{>}{\Theta})$ using **SLSA** 129

Preface

> `Magic, Can bring man all these and more---It can bring the universe to man, Whether seen or unseen.`
>
> — **J. Willrich Lutalo**, *Shrines of The Free Men*, 2018[6]

We are living in a world where man affects and influences nature so much, especially with his *unnatural* inventions in the form of animated and inanimate intelligence, implements and reality augmentations. But also, the world around man is constantly changing, and sometimes, in ways other than man is naturally comfortable with. That said, there is this idea that *a life unexplored isn't worth living*[1], and come what may, with every passing day, and as seasons and times change, it seems like the compulsion for humans to have to leave their traditional home environments and explore life elsewhere — say, in foreign countries, in-/under and on the sea, *inside* the earth or away from it, say like in outerspace, etc. seems so unavoidable — in fact, it feels to me like it is inculcated into our very natures, so that, thinking about a future in the same exact space, with the same exact people, environment and/or circumstances as we wake up to every day just doesn't make much sense[2]. That we must prepare ourselves — "we", meaning *humans*, to face an uncertain future, but also, to lessen such uncertainty to manageable levels where possible or necessary, is much of what justifies our investments in and explorations of science and knowledge in general.

For those who trust *divinely inspired* wisdom[3], the sacred scriptures (in **Genesis 1:28**) empower us thus:

> `God blessed them, saying to them,` *`Be fruitful, multiply, fill the earth and subdue it. Be masters of the fish of the sea, the birds of heaven and all the living creatures that move on earth.`*

Despite not having been funded by any one or any particular organization, the author, passionate about his pursuit of a **Doctorate in Philosophy** — of

[1] I trust, it is attributed to ancient philosopher and student of **Socrates**, Aristocles son of Ariston aka. 'Plato'.

[2] I have read a sane does of **Daniel Defoe**'s classic, *The Life and Adventures of Robinson Crusoe*, a relic from 1719, and among things Defoe seemed to argue against, was the somewhat strong urge in some humans, to want to go "abroad upon Adventures, to rise by Enterprise, and make themselves famous in Undertakings of a Nature out of the common Road".

[3] I keep an indispensable copy of **The New Jerusalem Bible**, and wouldn't wish to do much without consulting it first!

Computer Science or Mathematics, from the **University of Oxford**[7][8] or any other reputable institution[4], and especially to secure a teaching or faculty position, somewhere[5], embarked on this project, especially to further his earlier work on Transformatics[9] as well as to bring to better audiences, the now stable advances in his original, commercially-viable **TEA programming language**[10], by exploring new ground, proposing ideas and applications, in a domain potentially more in-demand and with far-reaching consequences for not just humanity, but all of nature, now, tomorrow, and in the deep, far-future.

[4]Refer to **ORCID:** `https://orcid.org/0000-0002-0002-4657`
[5]Because, in all fairness, knowledge must not only be sought or furthered, but must also be shared.

Abstract

Though it started out as just a paper, this work (now a monograph) brings to surface the importance of leveraging the mathematical theory recently named "Transformatics"[9], that deals with the study, processing and analysis of especially ordered sequences of symbols and their transformations. It has been demonstrated to be a credible and powerful theory in designing, specifying or explaining the properties of automata operating on sequences to produce other sequences — so-called **sequence transformers**. So, in this particular work, we take that body of knowledge, as well as what we know about the critically important science of how biological life-forms get to be defined, transformed and expressed in nature via the special genetic code known as DNA (deoxyribonucleic acid), that is best modeled conceptually as an ordered sequence of genes or more technically, a sequence of special combinations of any of four chemical bases (nucleic acids) or just "nucleobases" — Adenine(A), Cytosine(C), Guanine(G), and Thymine(T), into a finite sequence typically expressed as a double-helix ladder structure, that can then be decoded by convenient biological mechanisms so as to express or rather, manufacture one or more essential life-building compounds such as proteins that underlie the synthesis of specialized aspects of the organism's body such as skin, bone and muscle tissue in an animals or into cell-wall tissue, photosynthesis machinery and others, for plant organisms. This work demonstrates how several ideas first compiled under the transformatics umbrella can be well applied in problems relating to general genetics; we for example see how to quantify how far apart or different any two organisms might be based on the anagram distance between their genetic codes, this likewise being applicable to also sub-sequences of DNA that might underlie the expression of specific biological machines, certain traits or particular functions. We also consider how to leverage the concept of the modal sequence statistic in analyzing not just how similar different DNA sequences might be in terms of their relative composition of the basic nucleobases, but also in terms of their relative composition at higher abstraction levels such as at the level of amino-acids or large n-gram subsequences. We finally consider the matter of how, by leveraging the idea that a modal sequence encodes a summary about some larger sequence or an entire population of them, that we can then approach DNA as though it were a special statistical summary just like the MSS from transformatics theory, and just like how complex sequences could be constructed from summary statistics via certain protraction and multiplication transformers as we have encountered in earlier work on transformatics, we use a thought experiment to demonstrate how DNA would be transcribed into both an mRNA-like structure and then which can be further translated into actual body structures that allow the organism to occupy space and appear or behave in a particular way. Though this work is significantly theoretical and mathematical, we anticipate that these discussions and the exposition shall inspire actual domain experts and other researchers interested in genetics, genetic engineering and related sciences to pick up and apply our transformatics theory and ideas in both theory and practice.

Keywords: Applied Mathematical Statistics, Transformatics, Artificial Statistical Intelligence, Information Processing, Ordered Sequences, DNA sequences, Genetic Code, Genetic Analysis, Abstract Machines. Genome Sequencing, Genome Expression

Chapter 1

Introduction

Figure 1.1: In the course of writing this work, an experiment to grow pineapples from the pineapple fruit tops of pineapples normally bought from nearby marketplace (**Garuga/Bugabo-Bukaaya**) yielded results! Here we see the first successful harvest of a pineapple from our home garden (at **Nuchwezi Research**). But also, this case inspired part of the discussion in **Chapter 10** about how in real genome expression systems, the organism expressed from genetic code somewhat still includes the genetic code in the manifested organism as some kind of affix — a prefix (in leaves for example) or suffix (roots), from which the original organism (or the same exact genome) might be later reproduced!

> Sex seems to have been invented around two billion years ago. Before then, new varieties of organisms could arise only from the accumulation of random mutations --- the selection of changes, letter by letter, in the genetic instructions. Evolution must have been agonizingly slow. With the invention of sex, two organisms could exchange whole paragraphs, pages and books of their DNA code, producing new varieties ready for the sieve of selection. Organisms are selected to engage in sex --- the ones that find it uninteresting quickly become extinct. And this is true not only for the microbes of two billion years ago. We humans have a palpable devotion to exchanging segments of DNA today.
>
> — **Carl Sagan**, *COSMOS*, 1981[11]

> The material basis of heredity is DNA, a ladder-like molecule which carries a message in the form of a 'four-letter' code, the letters being four chemical bases, each of which may occupy any rung in the ladder.
>
> — **The Oxford Companion to the Mind**[12]

In reality, we find that living, real organisms are influenced by genetics, their environment, bits of randomness and sometimes emergent behaviors that might not be readily captured by strict rules such as the genetic code of life. However, away from all possibilities, and focusing on what can be said of life expression via the genetic code known as DNA (the *deoxyribonucleic acid*) and RNA (*ribonucleic acid*), especially when applied to the vast spectrum of natural organisms on earth — from basic **prokaryotes** (single-celled organisms) such as the simple bacteria that actually have no nucleus[13], to **eukaryotes**; all the way from basic one-cell kinds such as amoeba[14] all the way to sophisticated creatures such as oak trees, vultures, dolphins and human beings! Talking of which, it might be important to set clear that though **viruses** are a kind of life-form[15], and yet, they are neither prokaryotes nor eukaryotes — especially because, fundamentally, they are merely some genetic code (DNA or RNA) "enclosed in a protein coat, lacking cell membranes or organelles"[15] — more technically they are **acellular entities**.

So, if we bring on-board ideas from **transformatics**[9] — a new mathematical statistics theory dealing with sequences of symbols or those of named structures and their processing as well as analysis, for example, if we take the concept of leveraging statistical measures to summarize essential properties of sequences such as DNA — say, with use of the modal sequence statistic (MSS), we find that, independently of, and without needing to first consult or worry about mainstream genetic code analysis or genetic engineering theory and mechanics, that we can say many useful things about genetic code sequences and that we might be able to break new ground or solve some otherwise still intractable problems concerning sequences of genetic code.

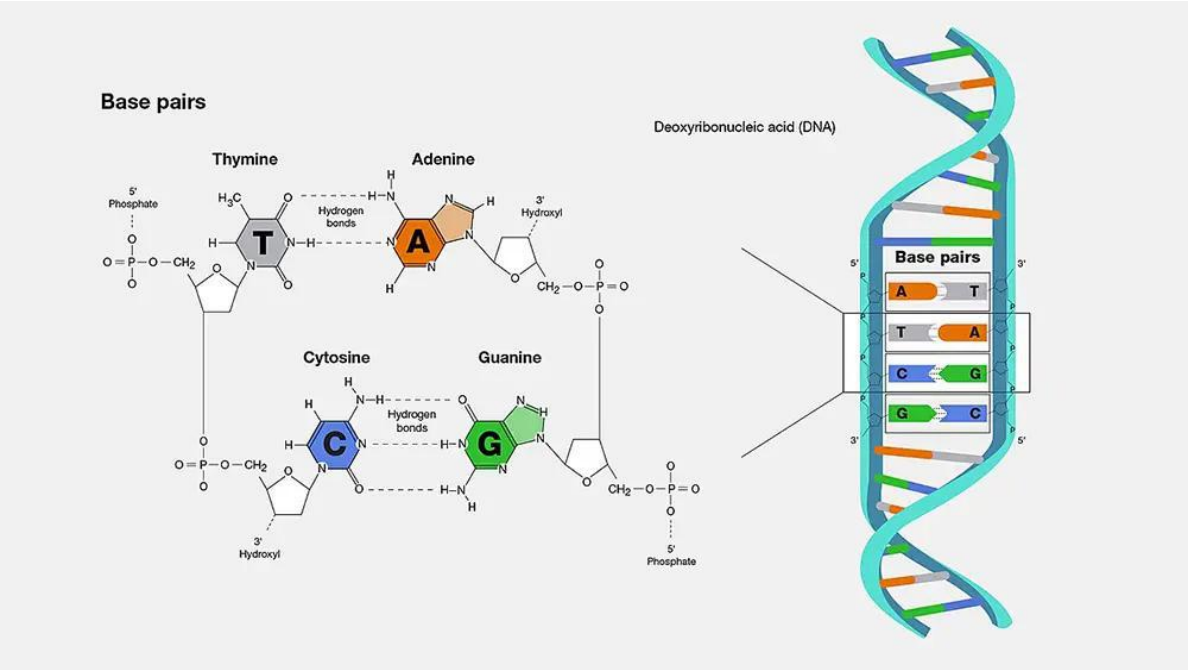


Figure 1.2: A basic diagram of the chemical structure of DNA base-pairs from [1]

For example, by borrowing a useful DNA-as-library metaphor from Venville et al[16], we would come to appreciate that by looking at *DNA as a library of books*, we have a model such as:

1. **Library**: DNA (as a whole) — the **Genome**, as the complete collection of genomic data about an organism.

2. **Bookshelves**: **Chromosomes** as organized storage of genes, and that they (chromosomes) are long, coiled-up strands of pairs of genes (essentially, the chromosome is a combination of two strands of genes, with one called the *template* and the other a *complement*, and that when they come together to form the "ladder" structure, at each step, the two genes forming a step are paired such that $A \leftrightarrow T$ and $C \leftrightarrow G$ [17] — also see **Figure 1.2**). So, for example, humans have 46 chromosomes in total in their "DNA library" (23 chromosome pairs, consisting of 22 autosomal pairs plus one pair of sex chromosomes). We know that during reproduction, the [full] genome (46 chromosomes in humans) splits up by half in either parent (via *meiosis*, which somewhat shuffles the complete genome and then halves it[18]), so that only one-half of the otherwise well-paired template-complement set that is the chromosome strand from each parent (as either sperm or ovum) goes to contribute to the final chromosome collection-set of the offspring[19][12]. Whereas, after fertilization or during normal cell division, the entire chromosome set (with well-paired strands) is duplicated/replicated wholesomely and losslessly so that the second/new cell thus created has an exact copy of the same chromosome set (entire genome) as was in the source cell[20].

3. **Books**: The **Genes** which make up a chromosome are the books in our genome library. Each gene is essentially a collection of "words" that are a sequence of one or more codons (see below), and which taken together, contain enough information/instructions to specify how to produce a specific protein[12][16].

4. **Words**: And then, under genes, we have **Codons** that are like "words" in each book, and each codon **only** codes for a single **amino acid**. Basically, a codon is a combination of any 3 of the "letters" of genetic code, for which, there are exactly four for DNA (A, C, G, T) and four for RNA (A, C, G, U). We also know that there are at most, 64 possible unique combinations of the four letters into triplets/3-grams/the codons[12].

5. **Letters**: Finally, at the most basic level of our genetic code/DNA-library/genome, we have "letters" that are technically known as **nucleotides**, and the nucleotide essentially each contains **a single nucleobase**, with only one difference between DNA and RNA as such; for DNA: $\{A, C, G, T\}$, and for RNA: $\{A, C, G, U\}$ — we have seen the names of the four letters for DNA, and the new one for RNA, 'U', stands for "Uracil".

So, with that introduction clearing up much of the basic genetics nomenclature and concepts that we shall use in the rest of this work, we then dive into the meat of our undertaking as laid out in the next section.

1.0.1 How to Navigate the Monograph

The necessary foundation concepts and nomenclature have been well introduced in **Chapter 1** — the previous section.

In **Chapter 2** we shall formally define the DNA and RNA symbol sets, ψ_{DNA} and ψ_{RNA}. We shall look at the matter of the difference between the ordering of terms in these mathematically-oriented sets, relative to the common-place **A-T-C-G** ordering typical in biological and especially genetics literature. We shall define the more universal ψ_{na} that would help apply logic to both DNA and RNA sequences, and shall generalize all such sequences as **na-Sequences**.

In **Chapter 3** we shall look at the idea of using n-grams such as the meaningful na-Sequence nucleobase 3-grams (technically known as **codons**) to abstract away various nuances of the flat-structure DNA or RNA sequence whose length might sometimes shoot into the millions or billions for realistic complete genome sequences. In **Chapter 9** we shall further these ideas when looking at how to process na-Sequences at n-gram level such as in naming or identifying subsequences. We shall also see that such abstractions might help manage complexity and simplify formalisms or algorithms dealing with na-Sequences such in the formalization of protein synthesis for natural or artificial automata via codon-processing ribosomes.

In **Chapter 4** we shall look at how we might leverage the basic concept of a symbol set to analyze arbitrary na-Sequences. We for example shall look at how to automatically generate special ordered symbol sets corresponding to individual na-Sequences under various bases — DNA, RNA, na- or even sometimes lexically meaningful AZ(ψ_{az}). We shall seen how to define special symbol sets based on n-gram abstractions of na-Sequences, such as might help in logic with special na-Subsequences such as START and STOP codons.

In **Chapter 5** we then shall consider the interesting matter of **complement sequences** and **complement symbol sets**. We shall look at how to generate complements given a starting sequence or symbol set. We shall look into the interesting fact that complementing ψ_{DNA} helps us arrive at the common **A-T-C-G** ordering of terms such as in $\dot{\psi}_{DNA}$. And though we might not use it immediately, we shall see that we might also talk of orthogonal symbol sets and orthogonal symbol set identities (o-SSI[21]) in relation to na-Sequences.

In **Chapter 6** we shall dive into how to leverage the **ADM** and **MSS** to perform non-trivial sequence analyses. We shall utilize four scenarios starting with basic scenarios between two individual sequences of potentially unequal lengths, all the way to analyzing entire collections or populations based on their representative na-Sequences. We shall look at how to leverage the **TEA**[22][10] text-oriented GPL to conduct some of the essential analyses proposed, readily via the the command-line on any major operating system, and especially with DNA/RNA sequences of any size stored in basic text files. We shall see that using the concept of the **modal sequence statistic**, whether applied to individual na-Sequences, collections of them or just sub-sequences of a longer DNA or RNA strands, can help us leverage statistical summary information about a sequence that might otherwise not be easy to glean or leverage from trivial inspection of a sequence. In this chapter too, particularly in **Section 6.6.1**, we shall develop an algorithm that might be utilized in applying statistical artificial intelligence to the matter of sequence classification problems relating to labeled datasets. The closely related **Section 6.6.2** shall consider the matter of identifying the **closest-sequence** from a collection of na-Sequences when

given a particular query na-Sequence. Finally, **Section 6.7** shall consider the important matter of **optimal database storage of na-Sequences** as is typically found in real-world systems such as those that do whole-genome sequencing or gene-banks, etc.

In **Chapter 7** shall get into some deep mathematics around the critical problem of genome sequencing. We shall tackle the fundamental problem in genetics engineering by breaking it down into 3 major problems. **Problem G1** dealing with the matter of how to compute the longest common subsequence given any two sequences. **Problem G2** dealing with how, as happens during conventional genome sequencing, we have sections of two or more sequences that are not exactly the same, but which must be reduced to some **consensus sequence** that is most true to all the input/available disparate identifying sequences. And then in **Problem G3**, we shall consider how to arrive at the final best consensus sequence that also unifies all the disparate input sequences originally provided, into a single sequence that is then the consensus genome sequence of the entity under analysis.

We shall then turn our attention to the matter of how genetic code is translated into actual living organisms in **Chapter 9**, focusing at the molecular level, and shall tackle the matter of the algorithm by which the most basic synthesis happens in a cell.

In the final chapter, **Chapter 10**, we shall look at what results from actual processing of genome sequences or rather DNA code; the manifestation of some living thing, an organism or some aspect of it as specified in the underlying genetic code. We shall especially deal with minimal entities such as viruses and simple organelles. Most importantly though, we we shall explore **genome expression** formally and conceptually so, with the assistance of a thought-experiment. A hypothetical genome-expression system — which we shall refer to as the **Lu-Genome Expression System** (LGES) shall be presented, and all our explorations shall be based on it. **Section 10.1** shall deal with a visual encoding method that leverages a digit-cipher named **OZIN**, and which is to be used to express storage-level genetic code (in decimal/numeric form), it its **intermediate form** — equivalent to DNA-to-mRNA transcription in living organisms. We shall finally turn attention to the final expression step — equivalent to mRNA-to-Protein/Function translation in nature, in **Section 10.2**, and shall get to see yet another innovation in the form of a genetic-code-to-spatial-form encoding based on a cipher we shall call the **Platonic Form Cipher** (**Section 10.2.1**). The essential algorithm for how to perform genome expression all the way from storage code into the final organism is depicted in **Algorithm 6**, and we shall close with some examples of complete, though hypothetical, fully-expressed genomes in **Section 10.2.3**.

Chapter 2

Sequence Symbol Sets and Sequence Transforms Applied to Genetic Code

NOTE:
On **Potentially Similar Mathematics**: *Frobenoids?*

Frobenioids[23], as introduced by **Prof. Shinichi Mochizuki**, offer a category-theoretic abstraction of arithmetic structures such as divisors and line bundles, encoding transformations via morphisms that resemble symbolic rewritings under algebraic constraints[24]. In spirit, they parallel the logic of sequence transformers (from transformatics[9]), which operate by mapping one symbolic sequence to another through rule-based transformations constrained by set-theoretic, statistical or general mathematical logic. However, even though both systems might serve as structured environments for encoding and navigating symbolic changes across mathematical objects such as sequences and/or sets, we just wanted to bring this up here, so students and researchers interested in our transformatics mathematics, and who wish to take the discussions and ideas we present in this work to even more advanced levels or in unconventional directions, pick a leaf and comfortably proceed to do so.

For the purposes of appreciating transformatics from a genetics engineering or general genetics research perspective, it shall be important to note that like in the original transformatics paper[9], beginning by appreciating that whatever formalisms and mechanics we might develop or talk about concerning genetic code or rather DNA, had better begin with an appreciation that we can model DNA as merely an ordered sequence of symbols.

In the introduction (see **Chapter 1**), we have already called out both the names and symbols assigned to the most basic units of any genetic code; essentially, the *nucleobases* (or rather, nucleotides). For purposes of simplifying our mathematical logic later on, we shall here neatly define what roles these units play in the grand scheme concerning DNA and RNA code. Basically, we shall want to define the symbol sets for any DNA sequence and the symbol set for any RNA sequence.

Definition 1 (The **DNA Symbol Set**, ψ_{DNA})**.** *For any possible sequence of standard deoxyribonucleic acid (**DNA**) for any possible living organism, the distinct nucleic acid base units are known as nucleotides[25], and these are essentially and exactly only four*[a]*;*

- *Adenine(A)*
- *Cytosine(C)*
- *Guanine(G)*
- *Thymine(T)*

And these are mapped to their representative, distinct single-letter symbols as shown. Thus, any possible DNA sequence must always consist of only one or more of those elements and nothing else. Thus, we might sum this up, using the symbol set concept[26] as applied to sequences[9] as such:

$$\psi(DNA) = \{A, T, C, G\} \tag{2.1}$$

Equation 2.1 *helps to appreciate the extra non-intuitive fact that the special ordering of DNA base symbols in the order A-T-C-G is what is conventionally accepted[27][25] or commonly found in most genetics literature*[b]*.*
*However, and especially because, for transformatics, we wish to work with an **ordered symbol set**[21] and not just any possible symbol set so that we can apply mathematical logic that respects the ordering of terms in any ordered sequence[9], we shall then assume a convention similar to how we might derive an ordered symbol set for a sequence of numbers in some base (the concept $\psi_{\beta}(\Theta)$ — see **Definition 5** in [21]), and given we are using Latin-Alphabet symbols (from ψ_{az}[9]) for ψ_{DNA}, we might as well better define the **Lexically Ordered DNA Symbol Set**, ψ_{DNA} as such:*

$$\psi_{DNA} = \psi_{az}(DNA) = \langle A, C, G, T \rangle \tag{2.2}$$

*For all practical purposes unless where we merely wish to emphasize adherence to the tradition of ordering the nucleotides by their pairing order, we shall essentially imply $\psi_{az}(DNA)$ or rather ψ_{DNA} when we talk of the **DNA Symbol Set**.*

[a] Actually, or rather, in general, for nucleic acid sequences, the bases are four for either DNA or RNA, but, there are also conventions that extend this set to 17 or more to cater for cases like where there might be ambiguity about what the exact nucleotide in a particular position might be[25].

[b] It shall be important to bring it out at this point, that, especially for non-domain experts — people not trained in or normally practicing in genetics or related fields, that the common ordering of the nucleotides in the A-T-C-G ordering might seem unconventional or peculiar! For example, one might wonder, why are they not listed in their alphabetical order? So, for purposes of settings things clear for everyone, the author consulted a reliable research assistant on this matter[24], and it was made clear that: "The order **A, T, C, G** isn't alphabetical, and yet it's the most commonly used sequence when referring to DNA bases." We further learn that, the order A-T-G-C reflects a mix of historical usage and bio-chemical structure; **base-pairing logic**: that the DNA's double-helix is stabilized by "complementary base pairing" in which A pairs with T (via 2 hydrogen bonds), and C pairs with G (via 3 hydrogen bonds), so that listing A with its partner T and then C with G emphasizes this pairing symmetry[24]. Further, we learn that early molecular biology texts and sequencing protocols (especially post-Watson & Crick, 1953) adopted this order to reflect the "functional relationships" between bases. It became entrenched in educational materials, sequencing software, and databases. And lastly, that in visual and structural conventions — such as in diagrams (see **Figure 1.2**) and models, A–T and C–G are often shown side-by-side. Listing them in this order reinforces the **duality** of the DNA ladder's rungs[24]. Lastly, that though there is no single documented moment when this convention begun, that the A-T-C-G order likely solidified in the **1970s - 1980s** during the rise of **Sanger sequencing** (developed in the 1970s), GenBank and EMBL databases, and overall in textbooks and molecular biology curricula.

Now that we have ψ_{DNA} and ψ_{RNA} well defined, we might immediately apply them to their finest use here: defining and supporting the special symbol set, ψ_{na}, that would or could allow for several kinds of nucleic acid sequences — such as DNA for code stored in chromosomes and mitochondria, various kinds of RNA (mRNA, tRNA,...) and even synthetic/artificial/conceptual and also **random nucleic acid sequences**, etc. using a single universal nucleic acid symbol set. Thus we define ψ_{na} below:

Definition 2 (The ***Universal Nucleic Acid Symbol Set***, ψ_{na})**.** *Any nucleic acid sequence,* Θ*, obeys the following law:*

Law 1 (**Nucleic Acid Identifier Symbol Set**: ψ_{na})**.** *The universal symbol set* ψ_{na} *spans any natural nucleic acid sequence* Θ*.*

Proof. $\psi_{na} = \psi_{DNA} \cup \psi_{RNA} = \langle A, C, G, T, U \rangle$ □

As with many kinds of sequences dealt with in transformatics, the possession of a particular symbol set, such as ψ_{na}, allows for the practical use of those sets to implement logic systems that can operate on signal based on symbolic expressions of well ordered elements in sequences or sub-sequences. For this matter then, we shall likewise want to make formal, the concept of a **na-Sequence**:

Definition 3 (The $na-$**Sequences** — (D/R)-NA Sequences: $\Theta_{na} : \mathbb{N} \times \psi_{na}$)**.** *Any sequence of DNA or RNA nucleotides — basically a sequence of nucleic acids, expressible as some sequence of symbols from* ψ_{na} *is a* ***na-Sequence*** *and its symbol set is a superset of both* ψ_{DNA} *and* ψ_{RNA}*. Equivalently:*

$$\forall \Theta_{na} = \langle a_i, \rangle, \quad a_i \in \psi_{na} \implies a_i \in \psi_{DNA} \vee a_i \in \psi_{RNA} \tag{2.3}$$

If it is not immediately clear what the significance of **Defition 1** is or why it's important to unambiguously define ψ_{DNA}, then perhaps the mathematical discussions in later sections like **Chapter 4** and **Chapter 10** might this more obvious. That said, since we know that nucleic acid sequences come in two flavors[25], and since we have covered the essential ground for DNA symbol sets, we need now also consider the ordered RNA symbol set.

Definition 4 (The **RNA Symbol Set**, ψ_{RNA})**.** *For any possible sequence of standard ribonucleic acid (****RNA****) for any possible living organism, the distinct nucleic acid base units are known as nucleotides[25], and these are essentially and exactly only four;*

- *Adenine(A)*
- *Cytosine(C)*
- *Guanine(G)*
- *Uracil(U)*

And these are mapped to their representative, distinct single-letter symbols as shown. Thus, any possible RNA sequence must always consist of only one or more of those elements and nothing else. Thus, we might sum this up, using the symbol set concept[26] as applied to sequences[9] as such:

$$\psi(RNA) = \{A, U, C, G\} \tag{2.4}$$

Equation 2.4 *helps to appreciate the traditional ordering of RNA base symbols in the order A-U-C-G reminiscent of the natural base-pairing order of DNA after it is translated into RNA (U merely replacing T)[28]. And as with DNA, we shall want to have a proper, meaningful* ***ordered symbol set****[21] for the RNA base symbols that we can later use in mathematical logic. Thus we shall equivalently define one for any standard RNA sequences as:*

$$\psi_{RNA} = \psi_{az}(RNA) = \langle A, C, G, U \rangle \tag{2.5}$$

And for all practical purposes in this work as well as after, we shall essentially imply $\psi_{az}(RNA)$ *or rather* ψ_{RNA} *when we talk of the* ***RNA Symbol Set*** *or the* ***Lexically Ordered RNA Symbol Set****.*

So, with those very essential definitions out of the way, we can begin to think of how we might appreciate and apply concepts from transformatics in genetics, in a clearer, straight forward manner.

For starters, we can certainly say that irrespective of how long or from where a particular DNA sequence, Θ_{DNA} originated from, once any sequence of data (e.g unstructured or unprocessed raw genetic code sequence dump), a string (say a proprietary encoding of some DNA code), a name (say of a particular species of interest and whose actual/representative genome sequence is know) or even a number (an id of a genome sequence in some standard genome database, etc.) is mapped to standard DNA sequence code, we shall know that any such transformation or mapping obeys **Theorem 1**:

Theorem 1 (ψ_{DNA} is the alphabet of any DNA sequence)**.** *For any sequence of DNA, Θ_{DNA}, produced from some or any other source Θ, irrespective of whether that source is itself a DNA sequence or not, we know that such an encoding or mapping, if it results in a valid DNA sequence Θ_{DNA}, obeys the following general transformation:*

Transformation 1. $\Theta \to \Theta_{DNA}$;
$\forall a_{i \in [1, \mathfrak{U}(\Theta_{DNA})]} \in \Theta_{DNA} \quad \exists \rho \in \psi_{DNA} : a_i = \rho$

Proof. Assume Θ_{DNA} contains some symbol α such that $\alpha \notin \psi_{DNA}$, it would contradict **Definition 1** which clearly states the legitimate membership of any DNA sequence. □

And definitely, the equivalent truth for RNA sequences would likewise follow from **Definition 4**. So, we can then correctly tell that a sequence such as $\Theta_{insulin}$, which is the genetic code sequence defined as:

$$\begin{aligned}\Theta_{insulin} = {} & \texttt{ATGGCCCTGTGGATGCGCCTCCTGCCCCTGCTGGCGCTGCTGGCCCTCTGGGGACC} \\ & \texttt{CCAGCCGCAGCCTTTGTGAACCAACACCTGTGCGGCTCACACCTGGTGGAAGCTCTCTAC} \\ & \texttt{CTAGTGTGCGGGGAACGAGGCTTCTTCTACACACCCAAGACCCGCCGGGAGGCAGAGGAC} \\ & \texttt{CTGCAGGTGGGGCAGGTGGAGCTGGGCGGGGGCCCTGGTGCAGGCAGCCTGCAGCCCTTG} \\ & \texttt{GCCCTGGAGGGGTCCCTGCAGAAGCGTGGCATTGTGGAACAATGCTGTACCAGCATCTGC} \\ & \texttt{TCCCTCTACCAGCTGGAGAACTACTGCAACTAG} \end{aligned} \tag{2.6}$$

And which sequence[1], despite having been obtained from an authority — the **Leiden Open Variation Database (LOVD)**, which is based on NCBI's RefSeq data[29], might need to be verified by hand or with a critical eye, and that's where a law such as **Theorem 1** would come in handy. Also, note that, unlike common sequence notations we might see in this work or that we have encountered before, we wrote $\Theta_{insulin}$ in a manner somewhat more convenient for the kind of verbose sequence that DNA code generally is. The notation — without opening or closing brackets around the sequence terms and neither commas in them might be reminiscent of the *String [Chart] Sequence* notation we developed recently (see **Transformation 15** in [9]).

We might for example start to wonder, how might we go about determining if some two DNA sequences, Θ_1 and Θ_2 are the same or perhaps if they share parts of each other, and by how much? We might wonder if they are the same sequence merely shuffled/anagrammatized — since we saw that this can occur during natural processes such as meiosis[18]. In case they are different, we might want to for example quantify the relative frequency of their distinct members, e.g mapping ψ_{DNA} or ψ_{RNA} to a sequence of relative frequencies, etc. These are all things we shall soon look into and know very well how to do or talk about, using genetics terms and the mathematical language and techniques from transformatics.

[1] $\Theta_{insulin}$ is the short coding DNA sequence (CDS) for the human **INS** gene — the part that gets transcribed into mRNA and translated by ribosomes into the insulin protein[24]

Chapter 3

Sequence Abstraction using Symbol Sets: From Flat-Structure Sequences to Higher-Level Sequences

Especially for nucleic acid sequences, but also for any other kind of sequence Θ, there might be legitimate situations in which looking at or dealing with the flat-structure sequence — such as for the actually modest[1] $\Theta_{insulin}$, might be cumbersome. And so, we are going to briefly look at ways that we might re-write verbose flat-sequences in more terse or higher-level abstract ways so as to focus on details that the flat-structure perhaps doesn't clearly express. This for example is naturally relevant in nucleic acid sequences since they are for example useful in scenarios where their processing is done in n-tuples/n-grams — such as with codons (nucleotide 3-grams) or as gene programs (self-contained sequences of codons), etc.

So, assuming we have some sequence of symbols Θ_1 defined as such:

$$\Theta_1 = \langle a_1, a_2, ..., a_n \rangle \tag{3.1}$$

If we wish to re-write the same sequence such that every k terms are grouped in the same subsequence, we can then re-write Θ_1 differently as such:

$$\Theta_2 = \Theta_2(n,k) = \langle a_{11}, a_{22}, a_{33}, a_{4k}, a_{51}, a_{62}, ..., a_{(n-k+1)(1)}, ..., a_{(n-k+k-1)(k-1)}, a_{(n-k+k)(k)} \rangle \tag{3.2}$$

Of course, in **Equation 3.2** we have somewhat wanted to extrapolate well and illustrate what would happen in an informative case such as when $k = 4$ and n is not only significantly larger than k, but is also its perfect multiple. Otherwise, we might more concisely write that same sequence as:

$$\Theta_2(n,k) = \langle \langle a_1, a_2, a_3, a_4 \rangle_1, \langle a_5, a_6, ..., a_8 \rangle_2, ..., \langle a_{(n-k+1)}, ..., a_{(n-k+k-1)}, a_{(n-k+k)} \rangle_{\frac{n}{k}} \rangle \tag{3.3}$$

So, this expression in **Equation 3.3** is our first proper appreciation of what higher-level sequences might be like when they are an abstraction of normal flat-structure sequences. We see for example here, that the $k = 4$ **bundling-parameter** means, we shall process the original flat sequence k items at a time, and so, each subsequence in $\Theta_2(n, 4)$ then contains exactly (or at most) 4 items — we are assuming 4 is a factor of n to keep things simple. And so, we expect that, in total, we should have $\frac{n}{k}$ subsequences after applying the bundling transform:

[1] 'Modest' because, genomes and realistic genome sequences can typically span millions of genes or billions of nucleotides. We for example know that for a typical human cell's DNA there are ≈3.2 billion base pairs[30], and yet we know each base pair consists of two nucleotides, one for each strand of the double-helix (so that's ≈6.4 billion), plus a few more (≈16,569 base pairs) from mitochondrial DNA[31]

Transformer 1 (The **k-GRAM Generator**). $\Theta = \langle a_1, a_2, ..., a_n \rangle \xrightarrow{O_{bundle-k}(\cdot)} \Theta^*$;
$\Theta^* = \langle \langle a_1, a_2, ..., a_k \rangle_1, \langle a_{k+1}, ..., a_{2k} \rangle_2, ..., \langle a_{(n-k+1)}, ..., a_{(n-k+k-1)}, a_{(n-k+k)} \rangle_{\frac{n}{k}} \rangle$
$\forall a_i \in \Theta \quad \exists a_{ij} \in \Theta^* : a_i = a_{ij} \quad \wedge \quad j \in [1, k]$
For some $n, k \in \mathbb{N}$*, and that* $\mathcal{U}(\Theta^*) = \frac{n}{k}$ = *number of k-gram subsequences generated from* Θ. □

Thus, we see that the sequence depicted in **Equation 3.3** with $k = 4$ could as well be produced by the generator defined in **Transformer 1**. But not just that, if we applied that transformer to the DNA sequence $\Theta_{insulin}$ first presented in **Chapter 2** — see **Equation 2.6**, we can then produce a proper standard codon/3-gram mapping of that DNA sequence that would look something like:

$$\Theta_{insulin} \xrightarrow{O_{bundle-3}(\cdot)} \langle \texttt{<ATG>,<GCC>,<CTG>,<TGG>,<ATG>,<CGC>,<CTC>,...,<ACC>} \\ \texttt{...,<TAC>,<TGC>,<AAC>,<TAG>} \rangle \quad (3.4)$$

And the way we have bundled up the nucleotides in **Equation 3.4** is exactly how a natural sequence processor such as a Ribosome (see details in **Definition 11**) would process it so as to produce say a corresponding protein.

Of course, though we might not delve into it here or at the moment, we know that by chaining several sequence transformers — such that one operates on the output of the previous one to produce the next sequence (see **Transformer 17** as an example), we can arrive at even higher abstraction sequences that might render certain sequence processing programs easier to write, analyze or define. For example, we know that, much as a Ribosome Processor operates on a sequence of codons, and yet, any legitimate protein synthesis and genetic code sequence processing program will require some sort of necessary delimiters (such as the START and STOP codons — refer to the Protein Synthesis Flow Chart in **Figure 9.2**), and so that, we might (if we could), abstract away codons and instead generate from a flat-structure DNA or perhaps mRNA sequence, a higher-level n-gram sequence of genes or **gene-program sequences**.

Chapter 4

Sequence Analysis Using Symbol Sets

We now turn our attention to the matter of using ideas from transformatics to help analyze or understand genetic code sequences. For starters, we shall want to consider the matter of how to tell how far apart or dissimilar any two nucleic acid sequences might be. Of course, consequently, that would also translate into meaningful way to tell how far apart not just low-level genetic-creations such as tissue, cell-types, proteins, etc might be based on the genetic code that produces them, but also how different entire organisms (within the same species or not) might be.

Because we can learn much about an organism based on its characteristic genome, and since that can be mapped to a flat-structure sequence of symbols such as we have already encountered in earlier sections, it might start to make sense to leverage the similarity/distance measures we have already developed so as to apply them to quantifying the distance between say species. In an earlier paper[32], we have introduced and also demonstrated how a sequence-analysis measure called the **Anagram Distance Measure**, $\tilde{A}(\Theta, \Theta^*)$, might be used to quantify how far apart some two sequences Θ and Θ^*, that for the simplest scenarios better be of the same length, can be compared based on their common membership and the relative lexical ordering of the members in either sequence.

In the simple analysis we shall use to illustrate or develop our concepts, we shall deal with hypothetical nucleic acid sequences mostly — especially DNA sequences, and their lengths shall be kept short to keep our analyzes simple, but also, their membership — in terms of nucleotides, might have nothing to do with actual/natural sequences. The concepts and ideas we shall develop though, should be readily applicable to any or most nucleic acid sequences if not any sequences in general.

First, assume we have the following three sequences:

1. $\Theta_1 = CATGGGACTGCC$
2. $\Theta_2 = ATAATAAGAGGGATCTGA$
3. $\Theta_3 = AUAGGGAGAAUC$

We can start by attempting to answer the question: **Which of these sequences are DNA and which are RNA?**

So, to answer that question, we can start by reducing each of those sequences to their respective [ordered] sequence symbol sets. In fact, if we first relax the assumption that they are either DNA or RNA sequences, and merely look at them as though they were any kind of sequence (for which we don't know the base or base-symbol set), then we can merely use the definition of an **Unspecific Symbol Set of Θ in Any Base** — see **Definition 4** in [21]. The algorithm for how to do that is simple and is well presented in that definition — we basically create another sequence, in which we insert each distinct symbol from Θ that we encounter while processing the sequence from Left-to-Right[1]. Thus, we might construct the relevant transformer for this as such:

[1]Such that the leftmost term is the first to be processed.

Transformer 2 (**Sequence Unspecified Symbol Set Generator**).
$\Theta \xrightarrow{O_{suss-gen}(\cdot)} \Theta^* \quad ; \quad \text{ш}(\Theta^*) \leq \text{ш}(\Theta) \quad \wedge \quad \Theta^* = \hat{\psi}(\Theta)$

And thus, applying that transformer to the three sequences we have, we shall obtain the following results:

Transformation 2. $\Theta_1 \xrightarrow{O_{suss-gen}(\cdot)} \langle C, A, T, G \rangle$

Transformation 3. $\Theta_2 \xrightarrow{O_{suss-gen}(\cdot)} \langle A, T, G, C \rangle$

Transformation 4. $\Theta_3 \xrightarrow{O_{suss-gen}(\cdot)} \langle A, U, G, C \rangle$

At this juncture, we can then closely inspect the resultant sequences — each of them a kind of symbol set, and then judge which of ψ_{DNA} or ψ_{RNA} they are associated with. To proceed in a rigorous manner, we might also want to compute the **Natural Symbol Set of Θ in some Base-β** — see **Definition 5** in [21]. A suitable transformer to compute such a symbol-set (for which, unlike the unspecific symbol set, orders the terms in the resultant sequence by their natural order of occurrence in the base's ordered symbol set) would as as such:

Transformer 3 (**Sequence β-Natural Symbol Set Generator**).
$\Theta \xrightarrow{O_{snss-gen-\beta}(\cdot)} \Theta^* \quad ; \quad \text{ш}(\Theta^*) \leq \text{ш}(\Theta) \quad \wedge \quad \Theta^* = \psi_\beta(\Theta)$

And, since we already have an idea what the symbol sets for each of the three sequences are — from which we can guess and/or disqualify some candidate bases — e.g, since $\hat{\psi}(\Theta_3) \backslash \psi_{DNA} = \{U\}$, then Θ_3 can't be a DNA sequence. However, to complete our analysis, note that:

Transformation 5. $\Theta_1 \xrightarrow{O_{snss-gen-DNA}(\cdot)} \langle A, C, G, T \rangle$

Transformation 6. $\Theta_2 \xrightarrow{O_{suss-gen}(\cdot)} \langle A, C, G, T \rangle$

Transformation 7. $\Theta_3 \xrightarrow{O_{suss-gen}(\cdot)} \langle A, C, G, U \rangle$

So, we can safely conclude that:

1. Since $\psi_{DNA}(\Theta_1) = \psi_{DNA}$, then Θ_1 is a DNA sequence.
2. Since $\psi_{DNA}(\Theta_2) = \psi_{DNA}$, then Θ_2 is a DNA sequence.
3. Since $\psi_{RNA}(\Theta_3) = \psi_{RNA}$, then Θ_3 is a RNA sequence.
4. Even if we forced it, note that $\psi_{RNA}(\Theta_1) = \langle A, C, G, T \rangle \neq \psi_{RNA}$, or rather that $\hat{\psi}(\Theta_1) \setminus \psi_{RNA} = \{T\}$ so Θ_1 can't be an RNA sequence.

Talking of which, in case we had some variation of Θ_1 that has no instances of T in it, such as the sequence $\Theta_4 = CAAGGGACAGCC$, then we shall find that:

Transformation 8. $\Theta_4 \xrightarrow{O_{suss-gen}(\cdot)} \langle C, A, G \rangle$

and that:

Transformation 9. $\Theta_4 \xrightarrow{O_{snss-gen-DNA}(\cdot)} \langle A, C, G \rangle \subset \psi_{DNA}$

But also that:

Transformation 10. $\Theta_4 \xrightarrow{O_{snss-gen-RNA}(\cdot)} \langle A, C, G \rangle \subset \psi_{RNA}$

And thus, we have the peculiar case in which a nucleic acid sequence can both be a legitimate DNA or RNA sequence! However, such a truly peculiar sequence it is! Because, given the START-codon symbol set, $\psi_{na-START}$, a sequence of all the "start" kind codons whether of DNA or RNA type, is defined as such from all we currently know:

Definition 5 (The **Nucleic Acid START-codons**)**.** *For any known organism, prokaryote or eukaryote, the only legitimate and known codons of the START-kind are any of the codons or START 3-grams in the unordered sequence $\psi_{na-START}$, the* ***nucleic acid START-codon symbol set****.*

$$\psi_{na-START} = \{ATG, AUG, GTG, GUG, TTG, UUG\} \tag{4.1}$$

So, that, if say a nucleic acid program for producing some protein via some gene program Θ_{na} were to be processed by an ideal ribosome (see ***Definition 11****), the ribosome would never produce anything — or, equivalently, there is no guarantee that any protein would be produced no matter what or any instructions the gene program contains, as long as it doesn't contain any one of the members of $\psi_{na-START}$.*

One interesting consequence of that definition is the following law:

Law 2 (**Non-Coding Gene Programs**[2])**.** *If any gene-program Θ_{na} is processed by a ribosome, and yet it has the property:*

$$\psi_{DNA}(\Theta_n a) \cap \psi_{na-START} = \emptyset \quad \vee \quad \psi_{RNA}(\Theta_n a) \cap \psi_{na-START} = \emptyset$$

It implies that Θ_{na} shall never be transcribed by the ribosome, and neither shall it ever result in any new protein or new amino-acid product within the containing cell system[3]*.*

So, that, if say a nucleic acid program for producing some protein via some gene program Θ_{na} were to be processed by an ideal ribosome processor (see **Definition 11**), the ribosome would never return — or, equivalently, there is no guarantee that any products — neither amino-acid, nor completed/new protein, shall be synthesized by the ribosome.

Of course, it's very likely that such gene sequences exist, and there might be nothing wrong with them being natural too. However, we shall want to learn more about this from the domain experts in the future.

That said, another, closely related case is that of the consequences of the STOP-codons or rather, the STOP-codon symbol set, $\psi_{na-STOP}$ defined as such:

Definition 6 (The **Nucleic Acid STOP-codons**)**.** *For any known organism, prokaryote or eukaryote, the only legitimate and known codons of the STOP-kind are any of the codons or STOP 3-grams in the unordered sequence $\psi_{na-STOP}$, the* ***nucleic acid STOP-codon symbol set****.*

$$\psi_{na-STOP} = \{TAA, UAA, TAG, UAG, TGA, UGA\} \tag{4.2}$$

Among important consequences of **Definition 6**, is that, if say a nucleic acid program for producing some protein via some gene program Θ_{na} were to ever be processed by an ideal ribosome (see **Definition 11**), the ribosome might produce something — some amino-acids for example,

[2]Also known as **Introns** in some contexts.
[3]Refer to ribosome definition for details: **Definition 11**

but would never return — equivalently, there is no guarantee that any protein would be released by the protein manufacturing process/ribosome, no matter what or how many any amino-acid productions and translate instructions the gene program contains and which have been executed by the ribosome. As long as the gene-program doesn't contain any one of the members of $\psi_{na-STOP}$ that is.

It is not immediately clear what the plausibility of such an *evil gene-program* existing out there in nature might be, but given normal genes sometimes mutate[12], it can't be ruled out that such awkward programs might exist. Though we won't dive into that here, who knows... from a computer security perspective, such a program might cause system errors or faults such as a System-Out-Of-Resources problem given the ribosome might attempt to produce an infinite length amino-acid, or perhaps that the cell's protein manufacturing closure might run out of space, or that the processor might end up hanging since the factory never is able to reach any of WAIT, DETACH or RESET states — see **The Ribosome State Machine** in **Figure 9.3** but also refer to the Protein Synthesis Process in **Figure 9.2** to clearly, logically appreciate these kinds of problems and/or corner-case scenarios.

That said, in case a normally correct gene-program such as what we saw in $\psi_{insulin}$ (see **Equation 2.6**) is altered with — perhaps by a natural mutation, or perhaps a virus, or even in a totally malicious feat of gene-manipulation medicine, and such a transformation results in a nucleic-acid sequence satisfying any of the above queer cases, who knows, but it might as well be the root case of some difficult flaw in the organism — an incurable and fatal disease (if the faulty gene-program is crucial for normal life support and that it has no alternatives or simple way to be solved), and these might the kinds of hard problems that might require not just medical doctors, biologists or perhaps genetists to tackle, but also people like computer hackers, information-security experts and software debuggers[4]!

[4]It shouldn't come as a surprise, that with a stronger marriage of computing theory and biology, or medicine, that certain problems in nature — not just with humans or animals, but also with plants for example, and especially if they are either inheritable — meaning genetic, or if they might somehow be remedied via clever hacking or debugging and modifying of the organism via gene programs, might call for the skills of and expertise of software engineers and program debuggers from hardcore computer science and not just the medical or biological sciences! A great background work on this subject by the author might come in handy — refer to [33]

Chapter 5

Sequence Analysis Using Complement Sets

The idea of complements when applied to na-Sequences has already been introduced in **Chapter 1**, and we get to see it depicted in an exemplary typical diagrammatic depiction of the DNA structure in **Figure 1.2**. However, apart from having talked about the fact that nucleic acids typically or naturally occur together in base-pairs depicting pairwise complements, we might want to leverage transformatics to study this idea at a more general level — say, to explore the matter of **na-Sequence** (as well as *any sequence*) complements, and thus this section.

Assume we start by exploring simple binary sequences — especially because bits help generalize many ideas in theory and practice. We can for example start by noticing that, like in the case of the interesting **Lu-Number System**[34] — in which any kind of *basic information* can be abstracted by mapping it to either signal (¿) or anti-signal (¡), and that *complex information* can then be generated from that using transforms on basic LNS sequences or expressions, etc. We can start by looking at what can be said of the transforms on the binary sequence symbol set, ψ_{bin} — or better, ψ_{01}. In our formulations, we shall denote the complement of a symbol ρ by $\neg\rho$, and for a sequence Θ, with $\neg\Theta$.

Sequence Name	Sequence	Formula	Note
$\Theta 1$	0 1	ψ_{01}	Binary Symbol Set as a basic Sequence
$\Theta 2$	0 1 0 1	$\psi_{01} \cdot \psi_{01}$	Sequence Concatenation/Multiplication
$\Theta 3$	0 2 0 1	$\psi_{01}(1 + \psi_{01})$	Sequence as Linear Combination of other Sequences: case of memberwise addition of the above two sequences
$\Theta 4$	1 0 1 0	$\neg\Theta 2 = \neg\psi_{01} \cdot \neg\psi_{01}$	Basic Complement Transforms

Table 5.1: An example of exploring symbol and sequence complements in sequence transformations

So, we can begin by noting that, if some sequence Θ spans some symbol set ψ_β — such as how all the sequences in **Table 5.1** span ψ_{01}, then we can say that the complement sequence, $\neg\Theta$ spans the complement symbol set $\neg\psi_\beta$. But not just that. You can readily tell from studying that table, that actually, if sequence Θ spans ψ_β, then it also spans $\neg\psi_\beta$ — meaning, its members essentially are one of the distinct elements from either symbol set even though the two sets might not necessarily have the same ordering of their members.

Further, note that, if ψ_β is the symbol set of some sequence Θ, and that the **complement symbol set** it is equivalent to is $\neg\psi_\beta$, then we can comfortably say that $\neg\Theta$ spans $\neg\psi_\beta$ as well. This is true, because, for any sequence Θ, its complement, $\neg\Theta$ is the sequence of the memberwise complements of Θ. That is to say:

$$\psi_\beta = \neg(\neg\psi_\beta) \tag{5.1}$$

and

$$\Theta = \neg(\neg\Theta) \implies \neg\Theta = \neg(\Theta) \tag{5.2}$$

So that, if

$$\psi_\beta = \langle \prod_{i=1}^{n} a_i \rangle \quad \implies \quad \neg\psi_\beta = \langle \prod_{i=1}^{n} a_{n-i} \rangle \tag{5.3}$$

And for any $\Theta : \mathbb{N} \times \psi_\beta$ we know that:

$$\neg\Theta = \prod_{i=1}^{n} \neg(a_i) \tag{5.4}$$

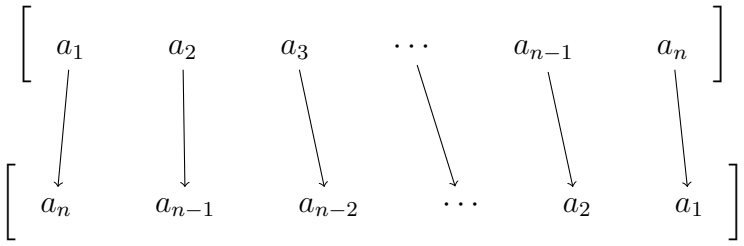
for $n = \text{ɯ}(\Theta)$ and that $\forall a_i \in \psi_\beta$,

$$\neg(a_i) = \rho_i \in \neg\psi_\beta \tag{5.5}$$

So, one way to appreciate **Equation 5.5**, is by realizing that by knowing the members and their order in ψ_β, we can then systematically determine the members and ordering of $\neg\psi_\beta$ via **Equation 5.3**, and so that, for some a_i in ψ_β, its complement in $\neg\psi_\beta$ is the term at position index $n-1$ in ψ_β. Thus, as we saw in **Table 5.1**, for $a_0 = 0$ in ψ_{01}, the complement, $\neg a_0 = a_{\underline{\nu}(\psi_{01})-0} = a_2 = 1$. This mechanics can be extended or applied to sequences of arbitrary length and composition as necessary. For purposes of future use, we shall generalize this with a useful theorem as such:

Theorem 2 (**Complement Sequences**). *For any ordered sequence* $\Theta_n\langle\prod_{i=1}^{n} a_i\rangle$*, its equivalent* ***complement sequence****, also denoted* $\neg\Theta_n$*, is the sequence with the signature* $\Omega_n\langle\prod_{i=1}^{n}\rho_i\rangle$ *such that* $\rho_i = a_{n-1} \quad \forall a_i \in \Theta_n$*. We shall also write* Ω_n *as* $\neg\Theta_n$ *where necessary.*

Proof. By **Equation 5.3**, we see that if $\Theta_n = \langle a_1, a_2, a_3, ..., a_{n-1}, a_n\rangle$, then we can derive its complement $\neg\Theta_n$ via the mapping of its members to corresponding members in its lateral inverse as such:



□

And since we are dealing with sequence transformations in this work, we might as well define the necessary transformer:

Transformer 4 (The **Complement Transformer**).
$\Theta \xrightarrow{O_{complement}(\Theta)} \Theta^*; \quad \Theta^* = \neg\Theta$

5.1 Significance of Sequence Complements, $\neg\Theta$, and Complement Symbol Sets, $\neg\psi_\beta$

First, we shall momentarily return to binary sequences.

We note that, for some binary sequence $\Theta : \mathbb{N} \times \psi_{01}$, the transform

Transformation 11. $\Theta \xrightarrow{O_{complement}(\cdot)} \Theta^*; \Theta^* = \neg\Theta$

which applies **Transformer 4**, produces the same sequence as Θ but with all bits swapped by their complements, which, as per usual nomenclature in computer science for example, means the resultant sequence is produced from the source via a **bitwise NOT operation**, and which, for binary sequences, is as simple as merely flipping the bits individually.

Thus, if our source sequence was

$$\Theta = \langle 01011110\rangle \tag{5.6}$$

Then

Transformation 12. $\Theta \xrightarrow{O_{complement}(\Theta)} \langle 10100001\rangle$

And then, returning to nucleic acid sequences, we note that if we for example have some na-Sequence such as

$$\Theta = \langle ATCGCGTAT \rangle \tag{5.7}$$

That we wish to treat as a DNA-sequence, then its *na-Complement* sequence would be derivable from ψ_{DNA} as such:

Since $\psi_{DNA} = \langle A, C, G, T \rangle$, then by **Equation 5.3**, we know that:

$$\neg(\psi_{DNA}) = \neg\psi_{DNA}) = \langle T, G, C, A \rangle \tag{5.8}$$

So that, $\forall \rho \in \psi_{DNA}$, the equivalent complement is via the mapping:

$$\begin{bmatrix} A & C & G & T \end{bmatrix}$$
$$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow$$
$$\begin{bmatrix} T & G & C & A \end{bmatrix}$$

Which is also the consequence of **Theorem 2**. Thus, for the sequence in **Equation 5.7**, the corresponding complement DNA sequence is:

$$\neg\Theta = \langle TAGCGCATA \rangle \tag{5.9}$$

This ideal can be further appreciated by noticing that the traditional DNA sequence base symbol set (which we shall denote by $\dot{\psi}_{DNA}$), is actually different from ψ_{DNA}, and is defined as:

$$\dot{\psi}_{DNA} = \langle A, T, C, G \rangle \tag{5.10}$$

We should come to appreciate this special symbol set, or even justify it — for logical and mathematical analyses, by acknowledging that it depicts not just a sequence of distinct symbols from ψ_{DNA}, but also their natural order based on both order of occurrence in ψ_{DNA} **and** their equivalent **symbol complements** in $\neg\psi_{DNA}$! It is such a terrific discovery/realization. And before we proceed, we note that the $\dot{\psi}_{DNA} \rightarrow \neg\dot{\psi}_{DNA}$ mapping for pairwise-ordered DNA sequences is as such:

$$\begin{bmatrix} A & T & C & G \end{bmatrix}$$
$$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow$$
$$\begin{bmatrix} G & C & T & A \end{bmatrix}$$

Because, from **Equation 5.10**, we can tell that:

$$\neg\dot{\psi}_{DNA} = \langle G, C, T, A \rangle \tag{5.11}$$

Perhaps, and worth checking out, even if momentarily, what would happen when the idea of complement symbols sets and complement sequences is applied to the more commonplace base-10 sequences (aka "usual numbers")?

First, note that since ψ_{10} — our choice of symbol for the decimal symbol set, is expressed as such:

$$\psi_{10} = \langle 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 \rangle \tag{5.12}$$

Then, by **Theorem 2**, its complement is:

$$\neg\psi_{10} = \langle 9, 8, 7, 6, 5, 4, 3, 2, 1, 0\rangle \tag{5.13}$$

So that, for any base-10 digit, the corresponding **complement digit** is as in the mapping below:

$$\begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \end{bmatrix}$$

Such a peculiar cipher! Because, usual/familiar number sequences such as the **Hi-Fi o-SSI**[32], defined as such:

$$\Theta_{HiFi} = 8649137520 \tag{5.14}$$

Has the complement:

$$\neg\Theta_{HiFi} = 1350862479 \tag{5.15}$$

Which, given how that special sequence was derived/discovered(see **Section 5.1** in [21] or **Equation 19** in the same paper), it is interesting to note that its complement under ψ_{10} is also an *orthogonal symbol set identity*! For those who have studied the o-SSI paper[21], this can really be exciting for the study of symbol sets and sequences in general — yes, even for nucleic acid sequences! But, that said, of course, it shouldn't come as a surprise, since, in simple terms, a sequence's complement is just one very special kind anagram among the set of all potential anagrams it can have, and so, before we leave the base-10 case, we can note that if some sequence $\Theta_{10} : \mathbb{N} \times \psi_{10}$ is a base-10 o-SSI, then its complement is also a base-10 o-SSI, and in fact, in general — even for non-numeric symbol sets such as ψ_{na}, ψ_{DNA} or special ψ_{RNA}, that:

Theorem 3 (**Complement of an o-SSI is an o-SSI**). *If some sequence $\Theta : \mathbb{N} \times \psi_\beta$ is an orthogonal symbol set identity for the base β, then its complement, $\neg\Theta$ is also an orthogonal symbol set identity for β.*

5.2 Potential Applications of Sequence Complements or Inverses

First, in genetics and analysis of genetic code sequences, note that for especially sequences in say the *non-native*(non-storage) form, such as the processed/derived RNA-sequences, it might sometimes be useful to determine if some two sequences are the same, equivalent or closely related, even though their apparent ordering of members, or even member composition looks unlike, **as long as their cardinalities are the same**[1].

We should note, and interestingly so, that the complement of any sequence Θ, which is $\neg\Theta$, is just an anagram of Θ, and that in general, we can comfortably say:

Theorem 4 (**Equivalence of Sequences under the Complement Transform**). *If two sequences Θ and Ω that are **not equal**, but which do have the same cardinality and similar distribution of members, then they are essentially equivalent under the sequence complement transform.*

Proof. The proofs are several:

1. Since $\text{ɯ}(\Theta) = \text{ɯ}(\Omega)$ and $\forall\alpha \in \Theta \quad \exists\rho \in \Omega : \alpha = \rho$ or equivalently, that $\forall a_i \in \Theta, \text{ɯ}(a_i \in \Theta) = \text{ɯ}(a_i \in \Omega)$, then there exists some anagram of Θ, denoted Θ^* such that $\Theta^* \xrightarrow{O_{complement}(\cdot)} \Omega$, which is possible by **Theorem 2**.

[1]This requirement to have their sizes the same, is a natural consequence of how a sequence and its anagram relate — see [32]

2. Because sequence complements are anagrams of each other, then if the two sequences are complements of each other, we expect that by the definition of the complement transformer (see **Transformer 4**), $\Omega = \neg\Theta$ or that $\Theta = \neg\Omega$.

3. We might also proceed by use of **modal sequences**(see **Definition 1** in [9]): Given the conditions on the two sequences, we then expect that if Θ is a sequence complement of Ω or vice-versa, then, computing their corresponding modal sequence statistic should result in the following result: $\overset{>}{\Theta} = \neg(\neg\overset{>}{\Omega})$ or equivalently, $\overset{>}{\Omega} = \neg(\neg\overset{>}{\Theta})$

□

Note that, for RNA, since $\psi_{RNA} = \langle A, C, G, U\rangle$ (**Equation 2.5**), then:

$$\neg\psi_{RNA} = \langle U, G, C, A\rangle \tag{5.16}$$

And this is what we'd expect via the complement mapping:

$$\begin{bmatrix} A & C & G & U \\ \downarrow & \downarrow & \downarrow & \downarrow \\ U & G & C & A \end{bmatrix}$$

But, since the traditional pairwise ordered symbol set for RNA, $\dot{\psi}_{RNA}$ is

$$\dot{\psi}_{RNA} = \langle A, U, C, G\rangle \tag{5.17}$$

Then its corresponding complement,$\neg\dot{\psi}_{RNA}$, is

$$\neg\dot{\psi}_{RNA} = \langle G, C, U, A\rangle \tag{5.18}$$

And so we have the associated complement mapping as

$$\begin{bmatrix} A & U & C & G \\ \downarrow & \downarrow & \downarrow & \downarrow \\ G & C & U & A \end{bmatrix}$$

So, that, for some RNA-sequence such as Θ_{met}:

$$\Theta_{met} = \langle A, U, G\rangle \tag{5.19}$$

Which is the famous START-codon **Methionine** (see **Table 9.2**), has as its corresponding nucleic acid sequence complement under $\dot{\psi}_{RNA}$:

Transformation 13. $\Theta_{met} \xrightarrow{O_{complement}(\Theta,\dot{\psi}_{RNA})} \langle G, C, A\rangle$
*That codon isn't the same as the source (ATG/AUG/Methionine), but there's some debate concerning what it actually is called — **Table 1** in [12] names codon GCA as **Arginine**, while Wikipedia[35] names it **Alanine**.*

While — **and important to note**, its complement under ψ_{RNA} is [indeed] different!

Transformation 14. $\Theta_{met} \xrightarrow{O_{complement}(\Theta,\psi_{RNA})} \langle U, A, C\rangle$
and we know that codon UAC is Tyrosine[35][2].

It should perhaps be immediately noticeable that the ability to derive new/different/special sequences such as complement codons from ordinary na-Sequences might have some useful if not exciting applications as well as problems it awakens for the geneticists, molecular biologists and researchers of gene-sequencing[3] and analysis. For example, apart from acknowledging that na-Sequence's complements denote potentially legitimate na-helix strand partners for any two sequences (in the form Θ and $\neg\Theta$), it would be interesting to determine what other important things might we learn, discover from exploring these kinds of sequence transforms?

It's already interesting to note that under complement transforms (which are special kinds of anagrams), the natural/universal protein synthesis START-codon AUG/ATG is equivalent to GCA (**Transformation 13**) or UAC (**Transformation 14**), but, **are those other codons also START-codons?** That said, is there a possibility that some natural process or genetic code translation or transformation might naturally produce ATG or AUG from any of those corresponding [symbolic] complement sequences? What would this teach us about natural/biological processors? What of the possibilities of using synthetic/artificial processors to render such complement operations useful?

[2]Concerning this particular codon, TAC/UAC, we find that though the Wikipedia reference tables assign it the name Tyrosine in both the DNA and RNA encodings, and yet, the physical reference we have at hand[12] (perhaps now out-of-date), assigns the name "Methionine" to TAC — Table 1 in that reference book didn't list RNA codes, but we still would expect to use TAC to look-up the correct name for UAC.

[3]For those that don't know: Gene sequencing is the process of determining the exact order of nucleotides — A, T, C and G — in a segment of DNA that makes up a gene. It's like reading the biological "code" that instructs cells how to build proteins and regulate functions[24].

Chapter 6

Sequence Analysis Using The Anagram Distance and the Modal Sequence Statistic

We are going to talk about the matter of comparing or analyzing two or more na-Sequences using especially statistical measures that have been set down by transformatics theory for especially ordered sequences.

6.1 Six Sequences and Four Scenarios

In particular, we shall be looking at na-Sequence analysis through lens of measures we have formalized earlier on, and shall break down our analysis into four representative scenarios:

1. **Scenario A:** Given some two particular instances of na-Sequences, $\Theta 1_n$ and $\Theta 2_n$, of the same length, n, and with similar symbol sets ($\psi_{na}(\Theta 1_n) = \psi_{na}(\Theta 2_n)$), how to quantify how similar or dissimilar they are?
2. **Scenario B:** Given some two particular instances of na-Sequences, $\Theta 1_n$ and $\Theta 2_k$, of different lengths, n and k, but with similar symbol sets ($\psi_{na}(\Theta 1_n) = \psi_{na}(\Theta 2_k)$), how to quantify how similar or dissimilar they are?
3. **Scenario C:** Given some two particular instances of na-Sequences, $\Theta 1_n$ and $\Theta 2_k$, of different lengths, n and k and with different symbol sets ($\psi_{na}(\Theta 1_n) \neq \psi_{na}(\Theta 2_k)$), how to quantify how similar or dissimilar they are?
4. **Scenario D:** Finally, how, when given a particular na-Sequence, Q_k, and two collections/samples of other sequences $(\Theta 1, \Theta 2, \Theta 3, ..., \Theta m)$ and $(\Omega 1, \Omega 2, \Omega 3, ..., \Omega n)$ of potentially dissimilar lengths, and that belong to labeled or classified populations — POPA and POPB, to determine which of the two populations the query sequence, Q_k best-belongs to (so we classify it under one of the labels), as well as determine which of the sample member sequences from either collections it is closest to?

Note that in conducting these analyses, we shall especially exploit the sequence-specific measures, $\tilde{A}(\cdot)$, the Anagram Distance Measure[32][9], and $\overset{>}{\Theta}$, the Modal Sequence Statistic[9], and that we shall use the following representative, even though somewhat hypothetical (for the first four), na-Sequences:

$$\Theta 1_{DNA} = \langle TACGGGCATTCC \rangle \tag{6.1}$$

$$\Theta 2_{RNA} = \langle AUGAUGCGCGGGCATAUC \rangle \tag{6.2}$$

$$\Theta 3_{DNA} = \langle ATAATGGGGGCATTGA \rangle \tag{6.3}$$

$$\Theta 4_{DNA} = \langle TACTCCGGGCAT \rangle \tag{6.4}$$

$$\begin{aligned}\Theta_{insulin} = {} & \texttt{ATGGCCCTGTGGATGCGCCTCCTGCCCCTGCTGGCGCTGCTGGCCCTCTGGGGACC} \\ & \texttt{CCAGCCGCAGCCTTTGTGAACCAACACCTGTGCGGCTCACACCTGGTGGAAGCTCTCTAC} \\ & \texttt{CTAGTGTGCGGGGAACGAGGCTTCTTCTACACACCCAAGACCCGCCGGGAGGCAGAGGAC} \\ & \texttt{CTGCAGGTGGGGCAGGTGGAGCTGGGCGGGGGCCCTGGTGCAGGCAGCCTGCAGCCCTTG} \\ & \texttt{GCCCTGGAGGGGTCCCTGCAGAAGCGTGGCATTGTGGAACAATGCTGTACCAGCATCTGC} \\ & \texttt{TCCCTCTACCAGCTGGAGAACTACTGCAACTAG}\end{aligned} \tag{6.5}$$

$$
\begin{aligned}
\Theta_{rbcl} = {} & AUGAAUUGCUUUGGAGGAGUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAG \\
& UUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAG \\
& GAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUG \\
& GAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCU \\
& UUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUG \\
& GCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCU \\
& UUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUU \\
& GCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAG \\
& UUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAG \\
& GAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUG \\
& GAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCU \\
& UUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUG \\
& GCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCU \\
& UUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUU \\
& GCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAG \\
& UUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAG \\
& GAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUG \\
& GAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCU \\
& UUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUG \\
& GCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCU \\
& UUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUU \\
& GCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAG \\
& UUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAG \\
& GAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUG \\
& GAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCU \\
& UUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUG \\
& GCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCU \\
& UUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUU \\
& GCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAG \\
& UUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAG \\
& GAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUG \\
& GAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCU \\
& UUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUG \\
& GCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCU \\
& UUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUU \\
& GCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAG \\
& UUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAG \\
& GAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUG \\
& GAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCU \\
& UUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUG \\
& GCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCU \\
& UUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUU \\
& GCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAG \\
& UUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAG \\
& GAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUG \\
& GAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCU \\
& UUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUG \\
& GCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCU \\
& UUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUU \\
& GCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAG \\
& UUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAG \\
& GAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUG \\
& \qquad GAGGAGUUUGCUUUGGCUUUGGAGGAGUUUGCUUUGGCUUUGGAGGAG
\end{aligned}
\tag{6.6}
$$

The first four sequences are merely hypothetical, and intentionally small/short, with $\Theta 1_{DNA}$, $\Theta 3_{DNA}$ and $\Theta 4_{DNA}$ being DNA-sequences, while $\Theta 2_{RNA}$ is just some RNA-sequence. $\Theta_{insulin}$ is a DNA-sequence we have already encountered in **Chapter 2** (see **Equation 2.6**), while the most verbose sequence we shall encountered in this work is named Θ_{rbcl} and is a real-life, biologically relevant RNA-sequence from a well-known plant protein: **RbcL**[1], the large subunit of **RuBisCO** — the enzyme responsible for carbon fixation in photosynthesis. It is one of the most abundant and essential proteins in plants[24].

6.2 A First Analysis

Before we dive into the matter of solving the analytic problems under the four scenarios introduced in **Section 6.1** above, we shall want to begin with a background analysis that shall give us some high-level insights about any of the sequences in our problem set. In particular, we are going to start by analyzing each of those sequences, so as to know the following basic facts:

1. What is the sequence's symbol set: $\psi(\Theta) =?$
2. Which of the three nucleic acid symbol sets does the sequence belong to — ψ_{DNA}, ψ_{RNA} or ψ_{na}
3. What is the relative frequency of each sequence's symbol-set members? $\forall \rho \in \psi(\Theta), \mu(\rho \in \Theta) =?$
4. How long is the sequence as a flat-structure sequence: $\mu(\Theta) =?$
5. How many codons does the sequence contain? $\Theta \rightarrow \Theta^* : \mathbb{N} \times \psi_{na}^3, \mu(\Theta^*) =?$

For purposes of helping others replicate the analyses and results we shall obtain, we recommend that the following analysis method be used: We can use the TEA[2][22][10] text-processing programming language — by the author, via its `tttt` Linux/Unix package to compute some of the analyses we wish to do via the command-line.

1. **To Process Sequence on Command-Line?** For example, copy-and-paste sequence $\Theta 1$ into a file named `theta1_dna.txt`
2. **To compute the sequence symbol set?** For example, for $\Theta 1$, run the following command against the sequence file to display the unique symbols in the sequence, in their natural order of first-occurrence within the sequence:

```
cat theta1_dna.txt | tttt -c "b:"
TACG
```

[1] Θ_{rbcl} is actually just a sample mRNA Sequence (partial, from *Arabidopsis thaliana* RbcL gene), and is excerpted from a paper that includes the *Arabidopsis thaliana* chloroplast genome, which contains the rbcL gene encoding the large subunit of RuBisCO[36].

[2] Instructions for how to install and use TEA/Transforming Executable Alphabet general-purpose text-processing oriented computer programming language are offered via the project's GitHub: `https://github.com/mcnemesis/cli_tttt`

3. **To compute the modal sequence statistic?** First, note that the **modal sequence statistic** is well introduced, defined and how to use it demonstrated in [9][3]. That said, for example, for Θ1, run the following command against the sequence file to display the unique symbols in the sequence, in their order of most frequent first, and/or their natural order of first-occurrence within the sequence:

```
cat theta1_dna.txt | tttt -c "u!"":"
CGTA
```

Note that the actual command/TEA-code for this is just "u!:", however, because of the way the Linux command-line treats the '!' symbol as a special character — even inside strings, we must cleverly use it in a manner such as shown; we write "u!" ":" to avoid the *"unrecognized history modifier"* error if that code is written directly on the terminal. This shall apply to all the other cases where we must deal with the TEA command qualifier '!"[22] The other workaround is to **first turn off the BASH history expansion modifier**, this, so that we can just write clean TEA code such as "u!:" without errors. So, to disable history expasion for the *active/current session* in your terminal, just run the command:

```
set +H
```

And after that, you can just write clean TEA code for the above task as such:

```
cat theta1_dna.txt | tttt -c u!:
CGTA
```

4. **To compute the distribution of symbols within a sequence?** For example, to compute how many times the symbol 'T' occurs in Θ1, we can merely run the code:

```
cat theta1_dna.txt | tttt -c "d!:T|g:|v:|v!:"
3
```

And for the symbol 'C', we just modify that code to count for 'C' as such:

```
cat theta1_dna.txt | tttt -c "d!:C|g:|v:|v!:"
4
```

But if we wish to do the same for all distinct symbols within the sequence Θ1 — **and especially without having to explicitly list which symbols we are counting within the TEA code**, we could have used "u!:" as in the previous task, since that TEA command counts how many times each unique symbol occurs in a sequence so as to compute the modal sequence, however, currently (with TEA version **1.0.8**[10]), the actual frequencies aren't

[3] See **Definition 1** in [9]

returned with the output of "u!:", and so, we shall want to use a work-around leveraging the above method for computing the frequency of each symbol in the input sequence. We can use the following non-trivial TEA program for that then:

```
cat theta1_dna.txt | tttt -c
  ↪ "v:vSEQ|v:vANA:{}|u!:|v:vMS|l:lMS|y:vMS|d!:^.|v:vSY|y:vMS
  ↪ |d:^.|v:vMS|y:vSEQ|d*!:vSY|g:|v:|v!:|v:vSYN|g*:{}:vSY:vSY
  ↪ N|x*:vANA|v:vANA|y:vMS|f:^$:lFIN|j:lMS|l:lFIN|y:vANA"
C4G3T3A2
```

The essential result from running that program against Θ1 is the string **C4G3T3A2**, which tells us that 'C' occurs 4 times, 'G' 3 times, ... , and then 'A' only 2 times.

5. **To compute the cardinality of a sequence?** For example, for Θ2, run the following command against the sequence file to display the total number of all symbols within the sequence:

```
cat theta2_rna.txt | tttt -c "v:|v!:"
18
```

To confirm if TEA is telling us the correct thing, we can also count the characters in that sequence (assuming we pasted just the sequence symbols into the file and nothing else, and no delimiters between them), using the standard Linux/Unix command "wc" as such:

```
cat theta2_rna.txt | wc -c
18
```

Such is the power of the sequence analysis we can do with the TEA language, without any sophisticated tools. Thus, shall we analyze all the other sequences in our problem set, and to simplify things, we shall merely tabulate the analysis results for all six sequences as such:

na-Seq	Unspecific Sequence Symbol Set: $\psi(\Theta)$	Closest Parent Symbol Set	Modal Sequence Statistic, $\overset{>}{\Theta}$ and Symbol Distribution: $\mathcal{U}(\rho \in \Theta)$	$\mathcal{U}(\Theta)$	Codon Count: $\mathcal{U}(\Theta^*) \approx \frac{\mathcal{U}(\Theta)}{3}$
$\Theta 1_{DNA}$	$\langle T, A, C, G \rangle$	ψ_{DNA}	C G T A 4 3 3 2	12	4
$\Theta 2_{RNA}$	$\langle A, U, G, C, T \rangle$	ψ_{na}	G C A U T 6 4 4 3 1	18	6
$\Theta 3_{DNA}$	$\langle A, T, G, C \rangle$	ψ_{DNA}	G A T C 6 5 4 1	16	$5.3\overline{3}$
$\Theta 4_{DNA}$	$\langle T, A, C, G \rangle$	ψ_{DNA}	C G T A 4 3 3 2	12	4
$\Theta_{insulin}$	$\langle A, T, G, C \rangle$	ψ_{DNA}	G C T A 108 105 60 56	329	$109.6\overline{6}$
Θ_{rbcl}	$\langle A, U, G, C \rangle$	ψ_{RNA}	U G A C 1355 1207 305 301	3168	1056

Table 6.1: A First Analysis of the 5 na-Sequences from **Section 6.1**

6.3 SCENARIO A — the anagram distance measure

`Given some two particular instances of na-Sequences,` $\Theta 1_n$ `and` $\Theta 2_n$`, of the same length,` n`, and with similar symbol sets (`$\psi_{na}(\Theta 1_n) = \psi_{na}(\Theta 2_n)$`), how to quantify how similar or dissimilar they are?`

So, in this scenario, we are basically given some two particular na-Sequences, of the same length, and which have similar symbol sets, and are tasked with how to quantify their similarity or dissimilarity.

Looking at our example sequences above in **Table 6.1**, only two sequences satisfy the necessary conditions — $\Theta 1$ and $\Theta 4$, which both have length **12** and which have similar symbol sets under base-na[4] — $\psi_{na}(\Theta 1) = \psi_{na}(\Theta 4) = \langle A, C, G, T \rangle$, but also their unspecific sequence symbol sets are already similar.

Actually, this case is very telling concerning what might actually happen with na-Sequences — essentially, we note that, despite the two sequences **actually being different** — for example, $\Theta 1$ ends with codon `TTC`, while $\Theta 4$ ends with `CAT`, and yet, looking at their breakdown analysis via **Table 6.1**, we see that **they almost seem to be the same!** They have the same sequence symbol set, the same parent symbol set, the same modal sequence and symbol distribution,

[4]Formally defined in **Definition 2**

the same sequence length and thus same number of codons! So, how exactly can we quantifiably distinguish between such sequences in real life?

At this juncture, it definitely might make sense to recall what the purpose of the **Anagram Distance Measure**[32] — it essentially is a measure that allows us to tell if any two sequences that might contain the same exact distinct symbols, but possibly in different proportions and/or order, are different or not. So, since we can't use the usually sufficient **modal sequence statistic**[9] to distinguish between these two actually different sequences, let us attempt to compute their anagram distance instead[5].

[5]It shall be **very important** to bring to mind here, the fact that, the cases where comparing sequences or datasets using the ADM might not seem obvious, but as we already saw in a detailed attempt to distinguish between actually different sequences using many various traditional statistical measures (of variation moreover) that all failed to show a difference between some two test sequences — see **Table 1** in [9], one shouldn't take ADM any less serious even for non-numerical data such as analysis nucleic acid sequences!

NOTE:
A Quick Recap **Concerning Interpreting ADM,** $\tilde{A}(\Theta, \Theta^*)$

The Anagram Distance Measure (ADM), associated testing method, background and justifications for the new statistic were first laid out in the seminal paper[32] introducing that measure and its theory. That work was later advanced in the trasformatics paper[9], and essentially, we can note that:

- Given any two sequences, Θ and Θ^*, that ideally should be of the same length, and with the same sequence symbol set, then we compute the anagram distance between them via the formula:

$$\tilde{A}(\Theta \to \Theta^*) = \frac{1}{\text{ш}(\Theta)} \times \sum_{i=1}^{\text{ш}(\Theta)} |I(\omega_i, \Theta^*) - i| \tag{6.7}$$

- And that we can interpret the results — which shall always be a positive real number ($\mathbb{R}^+$) as such:

$$\tilde{A}(\Theta \to \Theta^*) = \begin{cases} 0, & \Theta = \Theta^*, \text{no differences} \\ < 1, & \Theta \approx \Theta^*, \text{different in at minimum two positions} \\ = 1, & \Theta \neq \Theta^*, \text{all members shifted by exactly 1 position} \\ > 1, & \Theta \ll \Theta^*, \text{potential indication there is chaos.} \end{cases} \tag{6.8}$$

For a quick primer, on how this works out, assume we have $\Theta_1 = \langle a, b, c \rangle$, $\Theta_2 = \langle a, c, b \rangle$, $\Theta_3 = \langle b, c, a \rangle$, $\Theta_4 = \langle c, a, b \rangle$, $\Theta_5 = \langle b, a, c \rangle$. Then we can see that:

1. $\tilde{A}(\Theta_1, \Theta_2) = \frac{1}{3}(|1-1| + |2-3| + |3-2|) = \frac{2}{3} < 1$
2. $\tilde{A}(\Theta_1, \Theta_3) = \frac{1}{3}(|1-3| + |2-1| + |3-2|) = \frac{4}{3} > 1$
3. $\tilde{A}(\Theta_1, \Theta_4) = \frac{1}{3}(|1-2| + |2-3| + |3-1|) = \frac{4}{3} > 1$
4. $\tilde{A}(\Theta_1, \Theta_1) = \frac{1}{3}(|1-1| + |2-2| + |3-3|) = \frac{0}{3} = 0$
5. $\tilde{A}(\Theta_1, \Theta_5) = \frac{1}{3}(|1-2| + |2-1| + |3-3|) = \frac{2}{3} < 1$

However, to see the case when $\tilde{A} = 1$, we can use the sequences: $\Omega_1 = \langle a, b, c, d, e, f \rangle$ and $\Omega_2 = \langle b, a, d, c, f, e \rangle$. So, that we have:
$\tilde{A}(\Omega_1, \Omega_2) = \frac{1}{6}(|1-2| + |2-1| + |3-4| + |4-3| + |5-6| + |6-5|) = \frac{6}{6} = 1$
Concerning this last case, it might be worth noting that the only way to have that result for a sequence, is to have each element swapped with its immediate neighbor, without wrapping-around such as we saw happen in Θ_1 Vs Θ_3 for example. Also, the only way this could happen, is if the sequence's cardinality is even.

So, if we return to our na-Sequence analysis, we see that for our tricky case comparing Θ1 Vs Θ4, that if we compute their anagram distance, then we have the results:

$\Theta 4$	T	A	C	T	C	C	G	G	G	C	A	T
$\omega_i \in \Theta 1$	T	A	C	G	G	G	C	A	T	T	C	C
i	1	2	3	4	5	6	7	8	9	10	11	12
$I(\omega_i, \Theta 4)$	1	2	3	7	8	9	5	11	4	12	6	10
$\lvert I(\omega_i, \Theta 4) - i \rvert$	0	0	0	3	3	3	2	3	5	2	5	2

Table 6.2: A Tabular Analysis of Θ1 Vs Θ4 so as to compute their Anagram Distance

It shall be worth noting concerning how we derive the values of the updated position of ω_i in the compared/resultant sequence, $\Theta 4$, via $I(\omega_i, \Theta 4)$, that we are actually careful to follow the definition of that function (see **Note** on **Page 5** in [9]), with the useful detail that since both sequences contain repeated/duplicated symbols despite their similar length, that we assign to some symbol, such as the first 'T' in $\Theta 1$, the index of the first 'T' in $\Theta 4$, and second 'T' in $\Theta 1$, the index of the second 'T' in $\Theta 4$ — no skipping ahead, no juggling them up[6].

And thus, we can see readily that:

$$\tilde{A}(\Theta 1 \to \Theta 4) = \frac{1}{12}(0+0+0+3+3+3+2+3+5+2+5+2) = \frac{28}{12} = 2.3\bar{3} > 1 > 0 \tag{6.9}$$

And so, despite the measures and first analysis in **Table 6.1** having shown that these two nucleic acid sequences didn't have any significant differences, and yet, using ADM as in **Table 6.2** and **Equation 6.9**, we find that they are appreciably significantly different! Their ADM being 2.3, it tells us the second sequence potentially has elements in the first sequence shifted to new locations in the sequence, potentially by more than just 1 position. Thus, despite all their other similarities, they do have some differences that we can measure and pin to some telling numbers.

6.4 SCENARIO B — the sequence characteric [statistic]

> Given some two particular instances of na-Sequences, $\Theta 1_n$ and $\Theta 2_k$, of different lengths, n and k, but with similar symbol sets ($\psi_{na}(\Theta 1_n) = \psi_{na}(\Theta 2_k)$), how to quantify how similar or dissimilar they are?

So, given the conditions of this scenario and our sample sequences as summarized in **Table 6.1**, we shall pick sequences $\Theta 1$ and $\Theta 3$, which both have the same sequence symbol set under base-na, and which have cardinalities 12 and 16 respectively.

So, good enough, we have already worked through some of the essential details of the analysis method in **Section 6.3**. However, for this scenario, we can sum up how to conduct our comparative analysis thus:

1. First, conduct tabular analysis so as to see how the two sequences contrast against each other by the dimensions:

[6]Thus, we can generally say for any two sequences, Θ and Θ^* under ADM analysis, that for computing the $I(\omega_i, \Theta^*)$ for terms from the source, Θ, even where ω_i occurs multiple times in either sequence, even if at different positions, that we assign the first instance of ω_i, the value of $I(\omega_i, \Theta^*)$ equal to the position of the first instance of ω_i in Θ^*, the second instance of ω_i, the position of the second instance of ω_i in Θ^*, etc. So as to properly/correctly compute the ADM, but also for any other cases in using the **position-index function** in say logic or formulations as we do in many scenarios concerning Transformatics.

(a) Their unspecific[7] sequence symbol sets.

(b) Their closest parent/containing/superset symbol set.

(c) Their modal sequence statistic.

(d) Their **sequence characteristic** — more about this soon.

(e) Their cardinality.

We have already seen how to do all the above analysis steps and why, in the First Analysis described in **Section 6.2**.

2. So, if none of the analyses in the first step help show the **quantifiable similarities** or **quantifiable differences** between the two sequences, then proceed to using the more subtle Anagram Distance Measure on them. How to do this we have already covered well in **Scenario A**.
3. In case the ADM also can't find any differences between the two sequences, then most likely, their only differences are **cosmetic** — such as simply naming the same sequence differently, and thus the two sequences under analysis can be considered to be **quantifiably similar**.

So, first, note that, from **Table 6.1**, that even though the two sequences we are looking at, $\Theta 1$ and $\Theta 3$ have the same **specific sequence symbol set**[8] (under base-na or DNA): $\psi_{na}(\Theta 1) = \psi_{na}(\Theta 3) = \langle A, C, G, T \rangle$, and yet, their **unspecific sequence symbol sets** are different!

So, their first difference is in their unspecific sequence symbol sets: $\psi(\Theta 1) = \langle T, A, C, G \rangle$ and yet $\psi(\Theta 3) = \langle A, T, G, C \rangle$. So, at minimum, we know that they have quantifiable difference in the ordering of their similar symbols within either sequence just by this analysis — especially because these symbol sets respect order of first occurrence within the sequence they summarize.

If we must continue to still analyze the two sequences, we can further note that they have the same closest[9] parent symbol set, ψ_{DNA}. And so, that's a similarity.

And then we come to the matter of their modal sequence statistics. For this case, we see that we have the values:

- $\overset{>}{\Theta}1 = \langle C, G, T, A \rangle$
- $\overset{>}{\Theta}3 = \langle G, A, T, C \rangle$

So, this alone can tell us that the two sequences differ quantifiably in terms of either their relative distribution of members/symbols, or that they differ in the order of first occurrence of their symbols at worst.

[7] The **Unspecific Symbol Set** concept is first introduced in **Definition 4** of the o-SSI paper[21]

[8] The concept of the **specific symbol set** is first introduced in **Definition 3** of [21]

[9] We say "closest parent symbol set" because, for several scenarios in analyzing sequences, one can find cases where two sequences have symbols that belong to more than one symbol set where that other symbol set is larger than the sequence symbol set. For example, the case of having to decide if a sequence such as "101" belongs to base 2, base 3, base 10 or even base-36! Or if we have the sequence "AUG" that might span the RNA-symbol set, but also ψ_{na}, and talking of which, we must note that in principle at least, all na-Sequences could also be classified as sequences of terms from the Latin-Alphabet — the symbol set ψ_{az}!

Also, note that, the modal sequence statistic (MSS), is definitely computed readily by considering the relative frequency of terms within the sequence under analysis, and so, when we look at the MSS column in **Table 6.1**, we see that below each of the terms for the MSS, we also display the associated symbol frequencies. This is important as we are to see hereafter...

Talking of which, the next analysis we could conduct, but which is closely related to the previous one, is the **sequence characteristic**. Concerning this, note that, unlike **Scenario A** where the two sequences we analyzed had the same exact MSS, and yet, assuming $\Theta 1$ and $\Theta 4$ had some difference in the frequency of their terms despite having the same MSS, the MSS-analysis alone wouldn't have identified that. And so, before we proceed, let us also introduce/define the sequence characteristic measure.

Definition 7 (The **Sequence Characteristic**, $\hbar(\overset{>}{\Theta})$)**.** *If a sequence* Θ *has the modal sequence statistic* $\overset{>}{\Theta} = \langle \prod_{\omega_i \in \psi(\Theta)} \omega_i, \rangle$*, so that we can express it as a string concatenation of its distinct symbols in their relative order of highest frequency and first occurrence such as* $\omega_1\omega_2\omega_3 \dots \omega_k$ *for some* k *the size of* $\overset{>}{\Theta}$*, then, if for each* ω_i *we also know its corresponding frequency in* Θ*, such as* f_i *for* ω_i*, then writing the frequency next to the symbol such as* $\omega_i f_i$ *for all terms in* $\overset{>}{\Theta}$ *produces a string that contains both the information about the unique modal sequence of* Θ*, but also the information about how frequent each symbol occurred. We shall call such an expression the* ***Sequence Characteristic****, and shall denote it as* $\hbar(\overset{>}{\Theta})$*, so that we can then write for* Θ*, its corresponding sequence characteristic as:*

$$\hbar(\overset{>}{\Theta}) = \prod_{\omega_i \in \psi(\Theta)} \omega_i \cdot f_i \tag{6.10}$$

Closely related, and since we have already seen such a computation and its application in **Step#4** of our **First Analysis** using the TEA programming language, we might as well formally define a generic machine that can compute $\hbar(\overset{>}{\Theta})$ for any sequence.

Transformer 5 (The **Sequence Characteristic Generator**, gSC)**.**

$$\Theta \xrightarrow{O_{gSC}(\Theta)} \Theta^*; \quad \Theta^* = \hbar(\overset{>}{\Theta}) = \prod_{\omega_i \in \psi(\Theta)} \omega_i \cdot f_i = \prod_{\rho_i \in \overset{>}{\Theta}} \rho_i \cdot \mathbb{1}(\rho_i \in \Theta)$$

So, we note that, like in the example we saw in generating $\hbar(\overset{>}{\Theta}1)$ for $\Theta 1$ in our **First Analysis** — which returned the characteristic string as **C4G3T3A2**, we can then see that, by using the information in **Table 6.1**, that for the two sequences we are comparing, we have their sequence characteristics as such:

- $\hbar(\overset{>}{\Theta}1)$ = **C4G3T3A2**
- $\hbar(\overset{>}{\Theta}3)$ = **G6A5T4C1**

And thus, since those two measures aren't the same either, again, we have yet another evidence to conclude the two sequences are quantifiably different.

It should be worth noting, that in contrast to the previous scenario, the sequence characteristic[10] for $\Theta 1$ and $\Theta 4$ are equivalent (as expected?).

6.5 SCENARIO C

> Given some two particular instances of na-Sequences, $\Theta 1_n$ and $\Theta 2_k$, of different lengths, n and k and with different symbol sets ($\psi_{na}(\Theta 1_n) \neq \psi_{na}(\Theta 2_k)$), how to quantify how similar or dissimilar they are?

Without repeating ourselves nor wasting time, note that from the cases we have in **Table 6.1**, such might be the case with sequences such as $\Theta 1$ and Θ_{rbcl} for example. And, from the analysis procedure we have already encountered in **Scenario B**, we can definitely use any of several available ways to quantify their similarities or differences. However, most useful perhaps, might be to just look at their sequence characteristics:

- $\hbar(\overset{>}{\Theta}1)$ = **C4G3T3A2**
- $\hbar(\overset{>}{\Theta}_{rbcl})$ = **U1355G1207A305C301**

Which, automatically disqualifies any allegations that the two sequences are similar, and which, further tells us they differ in not only their symbol sets, but also in their symbol distribution.

Also, and important to note, we see that, by having a sequence's **characteristic**[11] computed, we can not only tell at a glance which symbols occur most frequent in a sequence, which symbols only occur once but in different order, but also, be able to readily compute the actual cardinality of the summarized sequence — so, with $\hbar(\overset{>}{\Theta}_{rbcl})$ = **U1355G1207A305C301**, we can compute the length of the **RbcL** by just summing up the f_i terms in $\hbar(\overset{>}{\Theta}_{rbcl})$: 1355 + 1207 + 305 + 301 = **3168**. Thus, the **sequence characteristic** statistic — see **Definition 7**.

[10] The **Sequence Characteristic** becomes yet another important contribution to the study and analysis of sequences that can serve special purposes than just the related Modal Sequence Statistic, and which, though derived from it, expresses different sequence summarizing information that is important in scenarios such as genetic sequence analysis as we have just seen. Definitely, this measure can, as with other measures we have developed in transformatics, be applied in any mathematical or scientific field and not just statistics or genetics such as in this case.

[11] Yes, sometimes we might just talk of a **characteristic** when we mean a "sequence characteristic".

Which is such a terrific measure when comparing especially large or arbitrarily sized sequences *at a glance*. Good enough, we have seen that there is a simple/short TEA program to automatically compute and display that measure for any input sequence, so, interested researchers and students can just adapt/build on that in the future.

6.6 SCENARIO D — the population characteric measure

> How, when given a particular na-Sequence, Q_k, and two collections/samples of other sequences $(\Theta 1, \Theta 2, \Theta 3, ..., \Theta m)$ and $(\Omega 1, \Omega 2, \Omega 3, ..., \Omega n)$ of potentially dissimilar lengths, and that belong to labeled or classified populations --- POPA and POPB, to determine which of the two populations the query sequence, Q_k best-belongs to (so we classify it under one of the labels), as well as determine which of the sample member sequences from either collections it is closest to?

So, unlike the other scenarios we dealt with, seems like the biggest concern here is to compare a sequence not against just one other sequence, but a collection of them — essentially, it's like having to quantify the distance between a particular sequence and some collection of other sequences. So how might we go about solving this using the tools already at our disposal?

NOTE:
Some Useful Ideas Concerning Classification Problems

Though we don't intend to borrow many or any ideas from what others would do or how existing [external and/or independent] methods might provide a solution to our classification problem, we shall call-out one useful general concept that is well placed in the Oxford Dictionary of Computing in relation to this kind of problem:

> **Decision Surface**: A (hyper) surface in a multidimensional state space that partitions the space into different regions. Data lying on one side of a decision surface are defined as belonging to a different class from those lying on the other. Decision surfaces may be created or modified as a result of a learning process and they are frequently used in machine learning, pattern recognition, and classification systems.
>
> — **Oxford Dictionary of Computing**[37]

First, let us deal with the matter of associating our query sequence, Q_k, one of several sequence sets or collections.

6.6.1 Measuring Proximity to a Set of Sequences, Ω^n

So, if we have a set of n different sequences of arbitrary composition and lengths, $\Omega^n = \langle \Omega 1, \Omega 2, \Omega 3, ..., \Omega n \rangle$, then, given some query sequence such as Q_k, and the consequences of **Theorem 2** in [9], we can safely classify the query sequence as **belonging to** or **being appreciably close** to the given population, if we find that computing the anagram distance between the query and the population via their representative modal sequence statistics results in an ADM value of 0, or if, where there are two or more populations to compare against, that we select that

population whose ADM against the query sequence's MSS is appreciably close enough to 0.

With that useful theory then, and given we already know how to compute an MSS, $\overset{>}{Q}_k$, for any particular sequence[12] Q_k, we instead better start by solving the matter of how to compute an MSS for a set of sequences — a **representative population modal sequence**.

So, assuming we have $Q_k = \langle a, d, c \rangle$, $\Omega 1 = \langle a, b, c, d, e, f \rangle$ and $\Omega 2 = \langle a, b, a, d, c \rangle$, and so that $\Omega^n = \langle \Omega 1, \Omega 2 \rangle$, we wish to compute $\overset{>}{\Omega}^n$, a population modal sequence statistic representing or summarizing the membership and frequency distribution across the two sequences. So, should we go about [correctly] computing $\overset{>}{\Omega}^n$?

STRATEGY 1: Since we have multiple sequences, and that they even have some uncommon terms and dissimilar lengths, a meaningful strategy might be to not attempt to compute the population MSS directly from the individual sequences, and instead use their sequence characteristics. That is, we would compute $\hbar(\overset{>}{\Omega}1)$, $\hbar(\overset{>}{\Omega}2)$, ... etc. and then using these, generate a combined **population characteristic**, $\hbar(\overset{>}{\Omega}^n)$. The essential definition shall help make things clearer:

Definition 8 (The **Population Characteristic**)**.** *Given a collection, Θ^n, of n sequences of arbitrary length and composition. The representative **population characteristic**, that summarizes all the sequences within that population, denoted $\hbar(\overset{>}{\Theta}^n)$, is derived from the sequence characteristics of the contained sequences, $\hbar(\overset{>}{\Theta}_1)$, $\hbar(\overset{>}{\Theta}_2)$, ... , $\hbar(\overset{>}{\Theta}_n)$ as such:*

$$\hbar(\overset{>}{\Theta}^n) = \prod_{\forall \omega_i \in \psi(\Theta^n)} \omega_i \cdot \mathit{ɯ}(\omega_i \in \Theta^n) \tag{6.11}$$

Where $\forall \Theta_j \in \Theta^n$

$$\mathit{ɯ}(\omega_i \in \Theta^n) = \sum_{\forall \hbar(\overset{>}{\Theta}_j)} f_{\omega_i} \tag{6.12}$$

And $\forall i, j \in \mathbb{N} : i < j \implies f_i \geq f_j$ for the frequency terms in $\hbar(\overset{>}{\Theta}^n)$.

Thus having computed the representative population characteristic for our sample/collection/set of sequences as $\hbar(\overset{>}{\Omega}^n)$, we then proceed to extract or reduce it to just the **representative population modal sequence statistic** (RPMSS)[13],

[12] In this work, check **First Analysis**, but also **Section 4.1** and especially **Definition 1** in [9] concerning how to compute the modal sequence statistic.

[13] Any careful analyst shall realize that computing or deriving the RPMSS as in the given process, or as per **Definition 6.11**, shall make the most sense, because, any other way would either ignore the important per-sequence symbol distribution information, or might just not be efficient or robust/trustworthy enough, plus, where the frequencies don't matter — such as when each symbol only

$\overset{>}{\Omega}^n$.

And with the RPMSS, $\overset{>}{\Omega}^n$, and having computed the MSS of the query sequence, $\overset{>}{Q}_k$, we can then proceed to compute the proximity between the query and the population via a measure such as the ADM: $\tilde{A}(\overset{>}{Q}_k, \overset{>}{\Omega}^n)$, and if there were more that one population to compare the query against, use the computed ADM values as a **decision surface** to help objectively decide which of the analyzed populations the query most likely belongs to.

Concerning this method too, it is important to note that given there might be cases in which the query sequence and the population contain several uncommon symbols — i.e. $\psi(Q_k) \setminus \psi(\Omega^n) \neq \emptyset$ or that $\psi(Q_k) \cap \psi(\Omega^n) = \emptyset$, we might need to adjust the modal sequences for both the query and population so that we can correctly compute their associated ADM. Thus, we might proceed by eliminating from both $\overset{>}{Q}_k$ and $\overset{>}{\Omega}^n$, those uncommon terms, so that the ADM we use as our decision surface is computed thus: $\tilde{A}(\approx \overset{>}{Q}_k \rightarrow\approx \overset{>}{\Omega}^n)$.

EXAMPLE:
How to Compute $\tilde{A}(\approx \overset{>}{Q}_k \rightarrow\approx \overset{>}{\Omega}^n)$

Assume we use the given sequences: $Q_k = \langle a, d, c\rangle$, $\Omega 1 = \langle a, b, c, d, e, f\rangle$ and $\Omega 2 = \langle a, b, a, d, c\rangle$, and so that $\Omega^n = \langle \Omega 1, \Omega 2\rangle$, then:

- $\overset{>}{Q}_k = \langle a, d, c\rangle$
- $\hbar(\overset{>}{\Omega 1}) = a1b1c1d1e1f1$
- $\hbar(\overset{>}{\Omega 2}) = a2b1d1c1$
- $\hbar(\overset{>}{\Omega}^n) = a3b2c2d2e1f1 \quad \Longrightarrow \quad \overset{>}{\Omega}^n = abcdef$

However, given $\overset{>}{\Omega}^n \setminus \overset{>}{Q}_k \neq \emptyset$ and/or $\overset{>}{\Omega}^n \cap \overset{>}{Q}_k = \{a, d, c\}$, then we must adjust as such:

- $\approx \overset{>}{\Omega}^n = acd$

And so that we can then compute $\tilde{A}(\overset{>}{Q}_k \rightarrow\approx \overset{>}{\Omega}^n) = \frac{1}{3}(|1-1| + |2-3| + |3-2|) = \frac{2}{3} < 1$.

If there is no other population to compare the query against, we can safely conclude the query is appreciably close enough to the population, otherwise we also compute the ADM between the query and the other populations, and then pick the one with the smallest ADM.

STRATEGY 2: The process outlined in the first strategy to solving the first problem in **Scenario D** might work well and be convincing enough for most practical and theoretical purposes, however, it is not the only plausible route to a good solution. Especially because we are mostly concerned with how to obtain the RPMSS for a collection of potentially widely dissimilar sequences — a

appears once across all sequences, then working from the per-sequence MSS would help generate a RPMSS that is close-enough to the relative ordering of terms in each member sequence.

very possible case if we are for example dealing with realistic collections of DNA sequence readings/scans of say a person's genome, the genome of some newly identified species with scanty details, or even the case of attempting to obtain an RPMSS for an entire set of individual organisms — e.g human members of a clan or particular family tree, etc. So, even though we might not delve into it here, there is a proposal to compute $\overset{>}{\Omega}^n$ from n sequences via computing the population's **genome sequence**.

This process would proceed somewhat like this:

- Using the provided collection of sequences Ω^n, and the process outlined in **Section 7.3**, especially via processing such a collection via the **Genome Sequencer** machine/transformer, we shall obtain a representative and summary, well-aligned and potentially complete [na-]sequence Ω_{gs} that represents the entire collection/population provided.
- By the **Identity Genome Sequence Law**, we trust that such a sequence shall be unique for any two distinct collections or populations.
- And thus, having obtained Ω_{gs}, we then compute the RPMSS not from the individual sequences in a population, but instead from their representative genome sequence. That is to say:

  Transformation 15. $\Omega_{gs} \rightarrow \overset{>}{\Omega}_{gs} \approx \overset{>}{\Omega}^n$

  And then we can use the obtained $\overset{>}{\Omega}^n$ and the query's $\overset{>}{Q}_k$ to determine how close they are via an anagram distance measure as we have already seen in earlier examples and analysis.

6.6.2 Measuring Proximity between a Q_k and Members of a Sequence Population, Ω^n

For answering the final aspect of **Scenario D**, we merely can proceed via the following Algorithm:

Algorithm 1 (The **Closest Sequence Algorithm**). *1. Compute the MSS for Q_k — i.e $\overset{>}{Q}_k$.*

2. *Since we are looking for the **sequence closest** to Q_k, then given we know $\overset{>}{Q}_k$, we also know $\psi(Q_k)$.*
3. *Initialize an empty collection Closest Sequence Tuples, **CST** := [].*
4. *Initialize **ADM_CST** := 0.*
5. *Initialize an empty collection Aproximately Closest Sequence Tuples, **ACST** := [].*
6. *Initialize **ADM_ACST** := 0.*

7. *For each sequence, Ω_i in Ω^n:*

   (a) *Compute $\psi(\Omega_i)$*

   (b) *Initialize **ADM_i** := 0.*

   (c) *If $\psi(\Omega_i) \setminus \psi(Q_k) = \emptyset$:*

      i. *Compute $\tilde{A}(\psi(\Omega_i) \rightarrow \psi(Q_k))$ and store that in **ADM_i***

      ii. *Iff **ADM_i** > **ADM_CST**, add tuple $[i, ADM_i]$ to **CST**.*

   (d) *Else/Otherwise:*

      i. *Adjust $\psi(\Omega_i)$ and $\psi(Q_k)$ to eliminate uncommon terms*

      ii. *Compute $\tilde{A}(\approx \psi(\Omega_i) \rightarrow \approx \psi(Q_k))$ and store that in **ADM_i***

      iii. *Iff **ADM_i** > **ADM_ACST**, add tuple $[i, ADM_i]$ to **ACST**.*

8. *If **CST** is not empty:*

   (a) *Sort **CST** in ascending order of the second term in each contained tuple: $[i, ADM_i]$ so that the tuple with the lowest ADM is the first in **CST**.*

   (b) *From Ω^n, return sequence with index in that topmost tuple as the solution.*

9. *Otherwise process **ACST**:*

   (a) *Sort **ACST** in ascending order of the second term in each contained tuple: $[i, ADM_i]$ so that the tuple with the lowest ADM is the first in **ACST**.*

   (b) *From Ω^n, return sequence with index in that topmost tuple as the solution.*

□

6.7 Concerning Optimal Symbolic Storage of na-Sequences in Databases - Computing the Gödel Number for a Sequence Θ_n based on its Characteristic $\hbar(\overset{>}{\Theta}_n)$

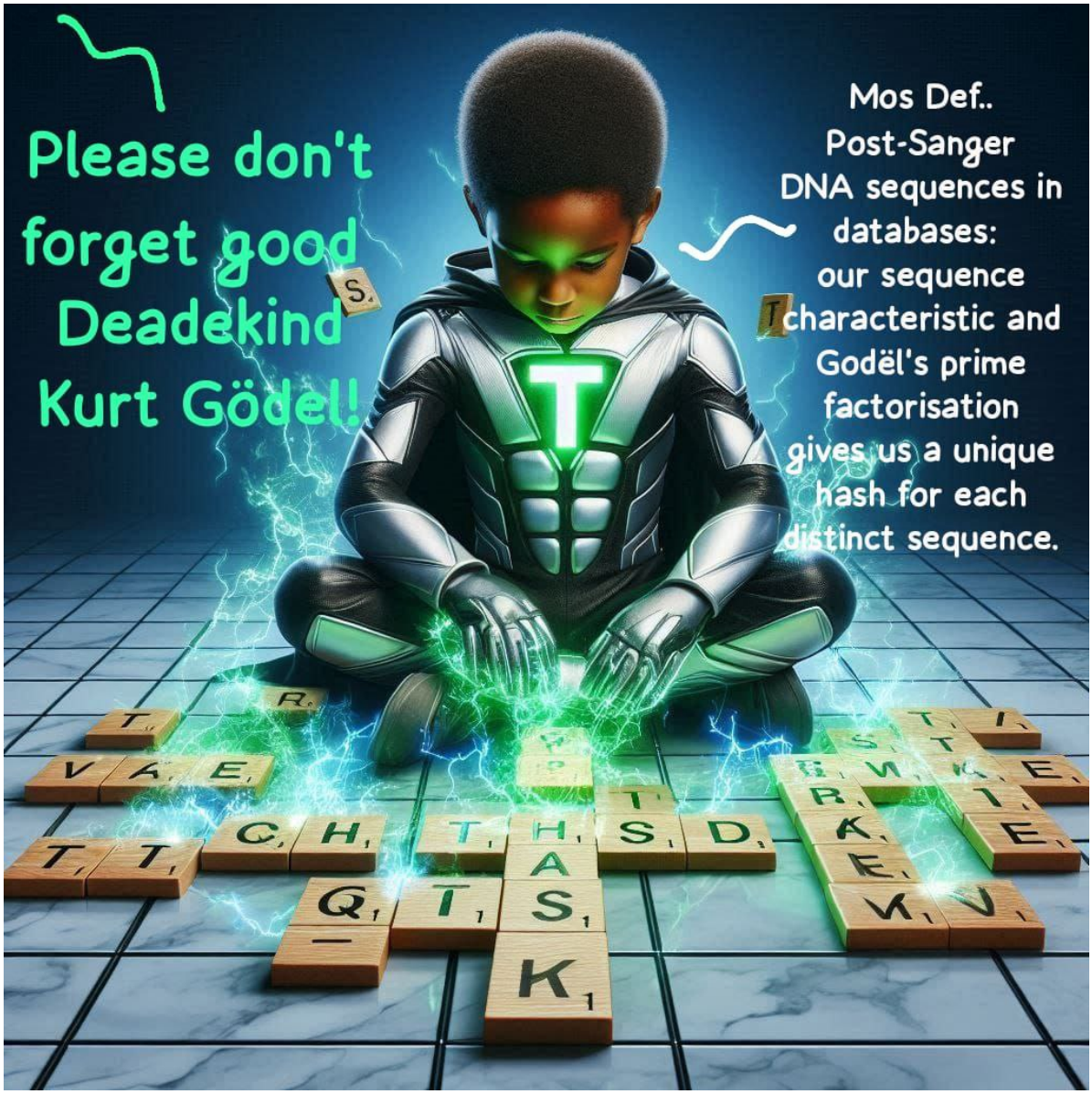


Figure 6.1: Some proposal to use Gödel's idea of abstracting formulae such as formal symbolic sequences using distinct natural numbers.

na-Sequences, for which we might have a collection of them, Θ^n, could have applied to them, a system of optimal or optimizable storage in practice, especially via electronic formats and protocols such as with flat text files, relative database systems and any well structured **operating memory space structures** — vaults in TEA[22], JSON-dictionaries on the Web[38], YAML files on local file-systems, etc. In this section, we could explore with the same na-Sequences introduced in **Section 6.1**, for ways how their corresponding optimal hash keys might be computed as part of an optimal-sequence lookup mechanism in bio-automata or

not.

Concerning Storage of an na-Sequence $\theta_n \in \Theta_k^n$ such that Θ has n sequences partitioned into k subsequences[14].

We then might wish to have some mechanism — such as the following generator:

Transformer 6 (A **Sequence Lookup System**, $\text{sLS}(Q_m, \Theta^n)$)**.** *:*
$\Theta^n \xrightarrow{sLS(Q_m,\Theta^n)} \theta_i$;

$$\text{ɯ}(Q_m) = m \leq \text{ɯ}(\theta_i) \quad \wedge \quad (\ddot{o}(Q_k) \approx \ddot{o}(\theta_i) \quad \vee \quad \tilde{A}(\overset{>}{Q}_k \to \overset{>}{\theta}_i) \leq \epsilon$$

□

By **Transformer 6**, we can search for and express or return, a complete na-Sequence $\theta_i \in \Theta^n$, when presented with some query na-sequence Q_m of length m.

Thus, for optimal lookup mechanisms — concerns about space and [polynomial] time for looking up and returning θ_{na} from some **na-database**, $D\langle \theta i_{key}, \theta i_{val} \rangle^*$, such that, instead of directly searching the collection for the sequence with value $\theta_i = \theta i_{val} \approx Q_m$, directly/naively using Q_m — the **user provided query**, we use the provably shorter — in terms of storage space, but also [*more intelligent-friendly* — in terms of sequence-expressing-information] higher **entropy encoding** structure, such as with a **hash key** for each θ_i as well as any Q_k, computed as:

$$\Gamma(id(\theta_i)) = \ddot{o}(\prod_{\omega_i, f_i \in \hbar(\overset{>}{\theta}_i)} \omega_i \cdot f_i) \approx \ddot{o}(\hbar(\overset{>}{R}_{\Theta_k^n})) \tag{6.13}$$

— see definition of $R_{\Theta_k^n} \approx \Theta_n$ in **IGS Law**, while the special sequence operator $\ddot{o}(\Theta_n)$ computes and returns a **Gödel Number**[37][15], $\ddot{o}(\Theta_n) : \psi_{\Theta_n}^k \times \mathbb{N}^k \to \mathbb{N}$ that is derived optimally using the characteristic, $\hbar(\overset{>}{\Theta}_n)$, for Θ a sequence of length n, and the first k prime numbers such that $k = \text{ɯ}(\overset{>}{\Theta}_n) \quad \wedge \quad \forall \omega_i \in \hbar(\overset{>}{\Theta}_n) : \ddot{o}(\omega_i) : \ddot{o}(i) = f_i \times p_i \in \mathbb{P}$ — p_i is the i-th prime[16] starting from 2 and spanning $\mathbb{N}$.

Computing a hashkey based on a Gödel number is just one approach to the optimal way to implement a sequence lookup mechanism with **Transformer 6**.

[14]This is just the proposed concept of the storage representation for an individual entire genome sequence, just one of many possibilities. Particular implementations might vary in actual sequence expression (in storage). For nucleic acid operating systems, we know that na-Sequences are in natural storage, when expressed as DNA (with partitions into nucleic acids, codons, genes, chromosomes, double-helixes of chromosomes, etc), however, while in use for processes such as RNA-oriented sequence processing (see **Section 9.1**, are operated on, not in bulk or whole-at-once, but in subsequences of some finite length — such as k in this formal specification, and $k = 3$ for typical, natural protein synthesis in ribosomes as an example.

[15]Based on the concept of **Gödel numbering** — "A one-to-one mapping (i.e. an injection) of the symbols, formulas, and finite sequences of formulas of the formal system onto some subset of the natural numbers. The mapping must be such that there is an algorithm that, for any symbol, formula, or finite sequence of formulas, identifies the corresponding natural number; this is the *Gödel number* of that object."[37]. Gödel numbers are a method developed by Kurt Gödel in 1931[39] to encode symbols, formulas, and entire proofs from formal mathematical systems as unique natural numbers. This encoding allows syntactic objects to be treated arithmetically[40].

[16]$\mathbb{P}$ is the set of all prime numbers.

However, even that method utilizes the transformatics concept of summarizing arbitrary sequences by their [proposed] **sequence characteristic** and consequently, their **modal sequence statistic**. However, as we see in the constraints and specification of that generator-transformer, the other option is to compute the solution using keys based on just the modal sequence statistics of the query and the target sequences in Θ^n.

Chapter 7

The Mathematics of Genome Sequencing: Sequence Alignment, The Modal Sequence, The Sequencing Machine and the Identifier Genome Sequence Law

Given what we know of genome sequencing theory and some problems we identified — such as an overall lack of proper mathematical formalism around the issue of how sequencing — or rather, computation of the resultant, consensus and most representative genome sequence (essentially, an identifying **na-Sequence**) of a living organism or particular aspects of it — proteins, mutations, behavioral traits, anomalies or bases of particular or general diseases or incapacities, etc. But even with the matter of mathematically expressing anomalous entities such as viruses and such, wasn't that compelling. And thus this section.

Seeing as the matter of determining the actual na-Sequence that correctly identifies a person or thing might be hinging on the case of attempting to reconstruct a book by mathematically piecing together disparate pieces of it — some in alignment, some not, some supersets of the others, some prefixes or suffixes of others, etc.[1] Also, though we might not immediately surface that philosophy here, the difficulties and problem thus raised, might likewise apply to the matter of trying to computationally "sequence" all of reality or even peculiar small aspects of it — sub-atomic particles that make up living things on the miniature scale, and then planets, stars and galaxies on the grand scale. Is it possible? Does it make sense? Is it necessary or even useful? These, and related problems are what we shall attempt to answer in the rest of this section.

[1] This analogy shouldn't come as a surprise, when compared to how we found it best to describe DNA using a figurative metaphor (see **Introduction**), but also based on how some authorities seem to be teaching this subject to their students[41].

7.1 PROBLEM G1: Computing MCS: Maximum Common Subsequence: $\Theta_1 \Diamond \Theta_2$

Problem 1 (*Computing MCS:* ***Maximum Common Subsequence***). *:* $\Theta_1 \Diamond \Theta_2$*] Given sequences* $\Theta_1 = \langle k, l, m, o, p, x, y, z \rangle$ *and* $\Theta_2 = \langle b, c, a, y, z, k, l, m, o, p \rangle$*, find the longest common subsequence they share — also to be denoted* $\Theta_1 \Diamond \Theta_2$*, their* ***M****aximum* ***C****ommon* ***S****ubsequence.*

Solution 1. *We shall specify a transformer, tMCS(*Θ_1, Θ_2*) that produces the required solution from the input sequences as such:*

Transformer 7 (The **tMCS** Transformer). $\langle \Theta_1, \Theta_2 \rangle \xrightarrow{O_{tMCS}(\Theta_1, \Theta_2)} \Theta_1 \Diamond \Theta_2$; $0 \leq \mathcal{U}(\Theta_1 \Diamond \Theta_2) \leq Max(\mathcal{U}(\Theta_1), \mathcal{U}(\Theta_2))$

□

Solution 1 is easier said than done. However, we shall attempt to formally and rigorously express it by formalizing what exactly the $tMCS(\Theta_1, \Theta_2)$ transformer does. Basically, we are to compute the MCS via **Algorithm 2**.

Algorithm 2 (The **tMCS** Algorithm). *1. Initialize the global maximum common sequence,* O_g^m*, to the empty set.*

2. *Given* ***a collection of*** n ***sequences****,* $\Theta^n = \{\Theta_1, \Theta_2, ...\}$.
3. *Start by computing the cardinality of each sequence,* $\Theta_i \in \Theta^n$*, and note the* ***largest sequence cardinality****,* $Max(\mathcal{U}(\Theta_i) \forall \Theta_i \in \Theta^n) = Max(\mathcal{U}(\Theta_i))$*, for the longest sequence,* Θ_m *such that* $Max(\mathcal{U}(\Theta_i)) = \mathcal{U}(\Theta_m)$.
4. *Compute the cardinality of a sequence that can exactly contain the given sequences each concatenated to the other, but with the longest sequence duplicated. Essentially, compute:*

$$C_R = \mathcal{U}(\Theta_m) + \sum_{\forall \Theta_i \in \Theta^n} \mathcal{U}(\Theta_i) \tag{7.1}$$

5. *Sort the collection of sequences,* Θ^n*, in ascending order of the sequence cardinality. Let the sorted sequence collection be denoted* $\hat{\Theta}^n$
6. *Start with the first two sequences from* $\hat{\Theta}^n$*, such as* Θ_1 *and* Θ_2*, and place them within two adjacent arrays* $A_1^{C_R}$ *and* $A_2^{C_R}$ *of equal length* C_R *— for simplicity, we shall just denote them as* A_1 *and* A_2*, in such a way that* Θ_1 *occupies in its containing array* A_1*, positions from 1 to* $\mathcal{U}(\Theta_1)$*, while* Θ_2 *occupies positions in the range:* $[(1 + \sum_{\Theta: \mathcal{U}(\Theta) > \mathcal{U}(\Theta_2)} \mathcal{U}(\Theta)) = I(\Theta_2[1] \in A_2)^2, I(\Theta_2[1] \in A_2) + \mathcal{U}(\Theta_2) - 1]$*, which also means,* $\forall a_i \in \Theta_j$ *such that* $I(\Theta_j, \hat{\Theta}_n) < I(\Theta_2, \hat{\Theta}_n)$*, it implies* $j < I(\Theta_2[1] \in A_2)$.

[2] We write $\Theta_2[1]$ to mean the first element in Θ_2 — i.e. position 1 has value 1, and the nth position has value n, not usual $n-1$

And Equivalently, the range $[\mathit{ш}(\Theta_1) + 1, \mathit{ш}(\Theta_1) + 1\mathit{ш}(\Theta_2) - 1] = [\mathit{ш}(\Theta_1) + 1, \mathit{ш}(\Theta_1) + \mathit{ш}(\Theta_2)]$.
Visually, we can express this as:

A_1	$a_{1,1}$	$a_{1,2}$	$\cdots$	$a_{1,\mathit{ш}(\Theta_1)}$						
A_2					$a_{2,1}$	$a_{2,2}$	$\cdots$	$a_{2,\mathit{ш}(\Theta_1)}$		

7. *Update the two sequence alignment arrays as such:*
   (a) *Keep the first array,* A_1*, unchanged (since it contains the longest sequence,* $\Theta_1 \in \hat{\Theta}_n$*).*
   (b) *Set* $j = z$*, where* z *is the* ***shift step size*** *— simplest scenario,* $z = 1$*.*
   (c) *Shift each element in the second sequence by* z*-steps to an earlier position in array* A_2*, so that the two sequences* Θ_1 *and* Θ_2 *have an overlap across the alignment arrays/matrix in positions spanning range:* $[\mathit{ш}(\Theta_1) + 1 - j, \mathit{ш}(\Theta_1)]$*, and so that* Θ_2 *shall be occupying the updated range:* $[\mathit{ш}(\Theta_1) + 1 - j, \mathit{ш}(\Theta_1) + \mathit{ш}(\Theta_2) - j]$
   (d) *Scan the overlapping sections in both sequences, in positions* $[\mathit{ш}(\Theta_1) + 1 - j, \mathit{ш}(\Theta_1)]$*, and by comparing each item in that range — i.e.* $\Theta_1[i]$ *Vs* $\Theta_2[i]$*, determine the longest common subsequence, or rather the* ***MCS****, thus:*

   Assume the overlap is of length m *such that the overlap can be visually depicted as such:*

$\approx \Theta_1$	$a_{1,k}$	$a_{1,k+1}$	$\cdots$	$a_{1,k+m}$
$\approx \Theta_2$	$a_{2,l}$	$a_{2,l+1}$	$\cdots$	$a_{2,l+m}$
$(\Theta_1[i] == \Theta_2[i])?$	*1*	*0*	$\cdots$	*1*
$\approx (\Theta_1[i] == \Theta_2[i]) \approx O^m$	$f(a_{1,k}, a_{2,l})$	*0*	$\cdots$	$f(a_{1,k+m}, a_{2,l+m})$

   for some $k \in [1, \mathit{ш}(\Theta_1)]$ *and* $l \in [1, \mathit{ш}(\Theta_2)]$*.*
   Create another array of maximum length m*,* O^m*, which contains at most* m *elements generated thus:*
   - $O^m[i]$ *contains 0 if the corresponding two elements in* Θ_1 *and* Θ_2 *didn't match. Otherwise it contains the matching element value from the first sequence — or rather,* $f(a_{1,k}, a_{2,l}) = a_{1,k}$ *iff* $a_{1,k} == a_{2,l}$*.*
   - *truncate* O^m *to its longest non-zero/non-empty subsequence, set that as* O^m*.*

- *Iff* $\mathcal{u}(O^m) > \mathcal{u}(O^m_g)$, *then* ***Update*** *or replace* O^m_g *with* O^m
  $\implies O^m_g := O^m$

(e) Update $j \implies j := j + z$

(f) Iff $j = \mathcal{u}(\Theta_2)$, *then return* O^m_g

(g) Otherwise Iterate from ***Step#c***

7.2 PROBLEM G2: Computing Overlap Resultant Sequences: $\Theta_1 * \Theta_2$

Problem 2 (*Computing ORS:* ***Overlap Resultant Sequence***). *:* $\Theta_1 * \Theta_2$*] Given two overlapping/well-aligned sequences* Θ_1 *and* Θ_2 *of the same length but potentially different terms at any position in the range* $[1, \mathcal{u}(\Theta_1)]$, *how to compute a resultant sequence* $\Theta_1 * \Theta_2$ *that is most meaningful for DNA sequences?*

Solution 2. *Assuming* $\Theta_1 = \langle \prod_{i=1}^{n} a_i, \rangle$ *and* $\Theta_2 = \langle \prod_{i=1}^{n} b_i, \rangle$, *construct* Θ^* *via the following tORS transformer:*

Transformer 8 (The **tMCS** Transformer).

$$\langle\langle \Theta_1 \langle \prod_{i=1}^{n} a_i, \rangle, \Theta_2 \langle \prod_{i=1}^{n} b_i, \rangle\rangle \xrightarrow{O_{tORS}(\Theta_1,\Theta_2)} \Theta_1 * \Theta_2;$$

$$\mathcal{u}(\Theta_1 * \Theta_2) = \mathcal{u}(\Theta_1) = \mathcal{u}(\Theta_2) = n$$

$$\wedge \quad \Theta^* = \langle \prod_{i=1}^{n} f(a_i, b_i), \rangle$$

for $f(x, y)$ ***a resolver function*** *defined as such:*

$$f(x,y) = \begin{cases} x, & x == y, \text{terms are the same} \\ x, & x = \neg(y), \text{one term is the complement of the other} \\ y & \text{otherwise.} \end{cases}$$

□

7.3 PROBLEM G3: Computing the Genome Sequence (GS): $(\Theta_1 \Diamond \Theta_2)^*$

Problem 3 (*Computing GS:* ***Genome Sequence***). *:* $(\Theta_1 \Diamond \Theta_2)^*$*] Given any two sequences,* Θ_1 *and* Θ_2, *compute their* ***resultant genome sequence****, denoted* $(\Theta_1 \Diamond \Theta_2)^*$, *that contains at core, their longest common subsequence (the MCS),* $O^m = (\Theta_1 \Diamond \Theta_2)$, *potentially padded either side by the specially computed prefix and suffix subsequences from either input sequence as such:*

$$(\Theta_1 \Diamond \Theta_2)^* \quad = \quad \Theta_1^a \cdot (\Theta_1^b * \Theta_2^b) \cdot (\Theta_1 \Diamond \Theta_2) \cdot (\Theta_1^c * \Theta_2^c) \cdot \Theta_2^a \tag{7.2}$$

And where the special terms are derived and named as such:

- Θ_1^a *is the non-overlapping section of* Θ_1 *before the* ***COS****.*

- $(\Theta_1^b * \Theta_2^b)$ *is the consensus sequence from* Θ_1 *and* Θ_2 *such that its elements are the result of applying the* ***resolver function*** *to compute* $f(\Theta_1[i], \Theta_2[i])$ *as specified in* ***Transformer 8****. It occurs* ***before the MCS****.*

- $(\Theta_1 \lozenge \Theta_2)$ *is the* ***maximum common subsequence****, MCS,* O_g^m*, of* Θ_1 *and* Θ_2*. It is the heart of the computed genome sequence.*

- $(\Theta_1^c * \Theta_2^c)$ *is the consensus sequence from* Θ_1 *and* Θ_2 ***after the MCS****.*

- Θ_2^a *is the non-overlapping section of* Θ_2 *after the* ***COS****.*

COS*,* $\overline{\Theta_1\Theta_2}$*, which is the* ***Consensus Overlapping Sequence****, padded on either by the non-overlapping sections of either* Θ_1 *or* Θ_2*, has the following properties:*

- $\overline{\Theta_1\Theta_2} = (\Theta_1^b * \Theta_2^b) \cdot (\Theta_1 \lozenge \Theta_2) \cdot (\Theta_1^c * \Theta_2^c)$

- $\Theta_1 \approx \Theta_1^a \cdot \overline{\Theta_1\Theta_2}$

- $\Theta_2 \approx \overline{\Theta_1\Theta_2} \cdot \Theta_2^a$

So that we could rewrite ***Equation 7.2*** *as such:*

$$(\Theta_1 \lozenge \Theta_2)^* \quad = \quad \Theta_1^a \cdot \overline{\Theta_1\Theta_2} \cdot \Theta_2^a \tag{7.3}$$

Finally, we know that the cardinality of the genome sequence, $\text{ɯ}((\Theta_1 \lozenge \Theta_2)^*)$*, is bounded thus:*

$$\text{ɯ}(\Theta_m) \leq \text{ɯ}((\Theta_1 \lozenge \Theta_2)^*) \leq \text{ɯ}(\Theta_m) + \sum_{\forall \Theta_i \in \Theta^n} \text{ɯ}(\Theta_i) \tag{7.4}$$

As per ***Equation 7.1*** *and the sensibilities of GTNC[26].*

7.4 The Genome Sequencer

Definition 9 (A **Genome Sequencer**). *A solution to **Problem 3**, is a special sequence processor, $\tilde{T}(\Theta^n)$, that can compute and return a resultant sequence, R_{Θ^n}, also known as the **genome sequence**, defined as*[a]

$$R_{\Theta^n} = (\Theta_m \Diamond \Theta^n \setminus \Theta_m)^* \tag{7.5}$$

Such that Θ_m is the longest sequence in Θ^n — a collection of n sequences, and that $\tilde{T}(\Theta^n)$ produces/generates/computes R_{Θ^n} by iteratively computing and updating the genome sequence in n-1 iterations as such:

Transformation 16. $R^0_{\Theta^n} \xrightarrow{O_{tGS}(\{R^0_{\Theta^n}\}\cup\Theta^n,0)} R^1_{\Theta^n} \xrightarrow{O_{tGS}(\{R^1_{\Theta^n}\}\cup\Theta^{n-1},1)} R^2_{\Theta^n} \xrightarrow{O_{tGS}(\{R^2_{\Theta^n}\}\cup\Theta^{n-2},2)} \ldots \xrightarrow{O_{tGS}(\{R^j_{\Theta^n}\}\cup\Theta^{n-j},j)} R^{j+1}_{\Theta^n} \ldots \xrightarrow{O_{tGS}(\{R^{n-2}_{\Theta^n}\}\cup\Theta^1,n-2)} R^{n-1}_{\Theta^n}$

*To produce the final genome sequence, $R^{n-1}_{\Theta^n}$ via the **tGS Transformer** defined as such:*

Transformer 9 (The **tGS Transformer**, $\tilde{T}(\Theta^n)$). $R^j_{\Theta^n} \xrightarrow{O_{tGS}(\{R^j_{\Theta^n}\}\cup\Theta^{n-j},j)} R^{j+1}_{\Theta^n}$; $R^0_{\Theta^n} = \emptyset$
and $R^{j+1}_{\Theta^n}$ is the resultant genome sequence after processing Θ^n with the first j sequences removed from it, and with the previous genome sequence, $\{R^j_{\Theta^n}\}$ appended to it, so as to produce that next genome sequence $R^{j+1}_{\Theta^n}$.

[a] Note that, even though in our formalism here we express the **relative complement set** of $\{\Theta_m\}$ in Θ^n as $\Theta^n \setminus \Theta_m$ — especially to keep the notation simple, we actually mean $\Theta^n \setminus \{\Theta_m\}$, which means, the first set with the elements in the second set omitted.

That transformer, $\tilde{T}(\Theta^n)$, is a genome sequencer, and is essentially a transformer that reduces a matrix of na-Sequences to a resultant sequence, R_{Θ^n}, that is the resultant genome sequence of a specie, population, or any natural entity represented by or expressed by the [potentially incomplete or approximate] genome sequences in the collection Θ^n.

7.5 The Identity Genome Sequence (IGS) Law

Law 3 (The **Identity Genome Sequence Law**). *The Identity Genome Sequence, $R_{\Theta^n}(\Omega) : \mathbb{N} \times \psi_{DNA}$, derived from some collection of sequences, $\Theta^n\langle\prod^n \theta_i\rangle$ such that $\theta_i \subset \Omega \vee \theta_i \approx \Omega$ are approximations of Ω, the longest known genome sequence of the entity, whose exact genome sequence would be Ω, and which can uniquely identify it. For sufficiently large n, and where the sample sequences θ_i were read or generated correctly by scanning or sequencing the entity, then*

$$R_{\Theta^n}(\Omega) \approx \Omega_m \tag{7.6}$$

Where Ω_m is the known best approximation of the genome sequence that correctly identifies any instance of a member of the kind Ω.

NOTE:
On **Consequences of IGS**

One potentially significant consequence of **Law 3** is that every distinct living thing — unique or distinct animal, plant, eukaryote or prokaryote, has a distinct identifying genome sequence, Ω, that we can discover and/or approximate via significantly many readings of particular or representative [sub-]sequences of the DNA of the observable expressions of Ω.
However, given that in reality it might not be exactly possible to exhaustively compute the correct value of R_{Θ^n}, nor tell exactly when that approximation approaches Ω exactly, it makes one wonder whether we can ever accurately know what it is that exactly underlies the expression of our reality — living things or not[a].

[a] Of course, there is the queer possibility that a kind of genome sequencing might be applicable to also inanimate/non-living things! But that's a philosophical discussion for another day.

Chapter 8

Random Sequence Generators (RSGs)

In many problems involving intelligent or non-linear processing of sequences, we might come across the need for either a random number, a random symbol or a random sequence of these, spanning some known/particular symbol set. This for example could be how we solve the problem: **Given the DNA symbol set ψ_{DNA}, how to generate a random DNA sequence of length n?**.

So, to help solve such critical and fundamental problems once and for all, and in a generic, transformatic-way, we shall specify some fundamental stochastic generator machines and algorithms, and these can then be used to solve any specific problem of generating a random sequence of a specific length given some finite symbol set.

8.1 The First Lu-Shuffle Algorithm (FLSA)

For this algorithm, for which we shall start by specifying its signature, the most important property it has is that it merely takes a sequence Θ as input and some parameter k — that **should lie in range** $1 \leq k \leq n$ and that $\text{ɯ}(\Theta) \geq 2$.

8.1.1 The FLSA Algorithm Using Mathematics

Algorithm 3 (The **First Lu-Shuffle Algorithm**: $\text{flsa}(\Theta, k_0)$)**.**

1. ***GIVEN*** *source sequence* Θ *of length* $n = \text{ɯ}(\Theta)$.
2. ***GIVEN*** *input parameter* k_0 *that is* ***Initial Partition Index.***
3. ***IF*** $n < 2$ ***RETURN*** Θ *unmodified.*
4. ***LIMIT*** *initial partition index using cardinality of input sequence, i.e*

$$k = k_0 \mod n \tag{8.1}$$

5. ***IF*** $k < 1$ ***UPDATE****:* $k = 1$.
6. ***INITIALIZE*** *the* ***resultant sequence****,* Θ^* *with* Θ.

$$\Theta^* = \Theta_{input} = \Theta \tag{8.2}$$

7. ***WHILE TRUE:***
   (a) ***BREAK IF*** $k = n$
   (b) ***SET*** *partition index,* $p_{index} = k$
   (c) ***SET*** *partition start,* $p_{start} = 0$
   (d) ***SET*** *sub-partition end/index,* $sp_{index} = \textbf{\textit{CEIL}}(\frac{p_{index}}{2})$
   (e) ***SET*** *left sub-partition,* $\Theta_l = \{\omega_i | \omega_i \in \Theta^* \quad \wedge \quad i \in [p_{start}, sp_{index}]\}$
   (f) ***SET*** *right sub-partition,* $\Theta_r = \{\omega_i | \omega_i \in \Theta^* \quad \wedge \quad i \in [sp_{index}, p_{index}]\}$
   (g) ***SET*** *unpartitioned sequence,* $\Theta_{rest} = \{\omega_i | \omega_i \in \Theta^* \quad \wedge \quad i \in [p_{index}, n]\}$
   (h) ***SET*** *swapped sequence,* $\Theta_{swap} = \Theta_r \cdot \Theta_l$
   (i) ***IF*** *k is* ***ODD****:* $(k \mod 2) = 1$*:*
      i. ***UPDATE resultant sequence,*** $\Theta^* = \Theta_{rest} \cdot \Theta_{swap}$
   (j) ***ELSE IF*** *k is* ***EVEN****:* $(k \mod 2) = 0$*:*
      i. ***UPDATE resultant sequence,*** $\Theta^* = \Theta_{swap} \cdot \Theta_{rest}$
   (k) ***INCREMENT*** *k:* $k = k + 1$
8. ***RETURN*** *resultant sequence,* Θ^*.

□

8.1.2 Examples of Applying FLSA

Using the basic, yet important sequence, ψ_{10} (refer to **Equation 10.1**) as the input, especially given we leverage such in many aspects of applied transformatics and genetics engineering in this work, we see hereafter, a trace of the accurate **invariant**[1] variants (actually, computed and predictable anagrams) of that set — all of them **base-10 o-SSIs**[21], when it is shuffled using **Algorithm 3** and various **meaningful values**[2] of the **initial partition index** k, starting from 1 and *preferably* up to $\frac{n}{2} = 5$ — we have also included values up to $n - 1$, just to drive the point home, that the algorithm shall return a result with some **k-grams**[3] of up to $k = \frac{n}{2}$ in the result for any $k > \frac{n}{2}$ — those large unshuffled subsequences might spoil the randomness of the resultant as you shall see when inspecting the examples we present.

[1] Basically, for a given particular input sequence such as ψ_{10}, we shall **always** get the same exact resultant sequence for a particular k, so that, it feels as though the algorithm randomizes the input sequence, but in a predictable/derivable manner, and this has its relevance in many problems as we shall come to appreciate.

[2] We say "meaningful" because, given how the algorithm works — shuffling the input sequence by partitioning it up into chunks and then swapping their order, the partitioning index, which determines where the swapping starts, **would not make much sense** for values greater than $\frac{\mu(\Theta)}{2}$, and for values less than 1. Also, the algorithm doesn't make sense for any value of k when $\mu(\Theta) \leq 1$. Luckily, the algorithm as specified in Algorithm 3 is designed so that it checks and applies the necessary remedies for meaningless values of k — such as setting $k = k \mod n$ for $k > n$ and $k = 1$ for $k < 1$. It returns Θ unmodified for cases where $\mu(\Theta) \leq 1$.

[3] The more common term is usually "n-grams"

```
GIVEN s=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
k=0 | s=[9, 7, 4, 0, 5, 3, 8, 2, 1, 6]
k=1 | s=[9, 7, 4, 0, 5, 3, 8, 2, 1, 6]
k=2 | s=[8, 6, 3, 9, 4, 2, 7, 1, 0, 5]
k=3 | s=[8, 6, 3, 9, 4, 2, 7, 0, 1, 5]
k=4 | s=[5, 3, 0, 6, 1, 7, 4, 8, 9, 2]
k=5 | s=[5, 1, 2, 6, 3, 7, 4, 8, 9, 0]
k=6 | s=[0, 8, 9, 1, 5, 2, 6, 3, 4, 7]
k=7 | s=[3, 8, 9, 4, 2, 5, 6, 0, 1, 7]
k=8 | s=[9, 1, 2, 3, 8, 4, 5, 6, 7, 0]
k=9 | s=[9, 5, 6, 7, 8, 0, 1, 2, 3, 4]
```

Figure 8.1: A Tabulation Trace of Shuffling the Base-10 n-SSI Sequence with increasing values of k, the **Initial Partitioning Index** for **Algorithm 3**, the **First Lu-Shuffle Algorithm**.

```
GIVEN s=[1, 2]
k=0 | s=[2, 1]
k=1 | s=[2, 1]


GIVEN s=[1, 2, 3]
k=0 | s=[3, 2, 1]
k=1 | s=[3, 2, 1]
k=2 | s=[2, 1, 3]


GIVEN s=[1, 2, 3, 4]
k=0 | s=[1, 4, 3, 2]
k=1 | s=[1, 4, 3, 2]
k=2 | s=[4, 3, 2, 1]
k=3 | s=[4, 3, 1, 2]
```

Figure 8.2: A Tabulation Trace of Shuffling basic base-case scenarios to help highlight how **FLSA** works.

NOTE that, after looking at the examples in **Figure 8.1** and **Figure 8.2**, one might somewhat start to comfortably say that the **best range** for the partitioning index k, is possibly $1 \leq k \leq \frac{n}{4}$.

We might also want to look at cases of generating shuffled or randomized na-Sequences based on well-known, useful starter sequences (not necessarily symbol

sets, or deduplicated sequences, but still useful). For example, Θ_{2na} from **Equation 10.27**

```
GIVEN s=['A', 'C', 'G', 'T', 'U', 'A', 'C', 'G', 'T', 'U']
k=0 | s=['U', 'G', 'U', 'A', 'A', 'T', 'T', 'G', 'C', 'C']
k=1 | s=['U', 'G', 'U', 'A', 'A', 'T', 'T', 'G', 'C', 'C']
k=2 | s=['T', 'C', 'T', 'U', 'U', 'G', 'G', 'C', 'A', 'A']
k=3 | s=['T', 'C', 'T', 'U', 'U', 'G', 'G', 'A', 'C', 'A']
k=4 | s=['A', 'T', 'A', 'C', 'C', 'G', 'U', 'T', 'U', 'G']
k=5 | s=['A', 'C', 'G', 'C', 'T', 'G', 'U', 'T', 'U', 'A']
k=6 | s=['A', 'T', 'U', 'C', 'A', 'G', 'C', 'T', 'U', 'G']
k=7 | s=['T', 'T', 'U', 'U', 'G', 'A', 'C', 'A', 'C', 'G']
k=8 | s=['U', 'C', 'G', 'T', 'T', 'U', 'A', 'C', 'G', 'A']
k=9 | s=['U', 'A', 'C', 'G', 'T', 'A', 'C', 'G', 'T', 'U']
```

Figure 8.3: A Tabulation Trace of Shuffling the na-Sequence Θ_{2na} from **Equation 10.27** with **Algorithm 3 (FLSA)**.

A closely related set of random na-Sequences would be that generated with **Algorithm 3** from the palindrome sequence Θ_{palna} from **Equation 10.28**:

```
GIVEN s=['A', 'C', 'G', 'T', 'U', 'U', 'T', 'G', 'C', 'A']
k=0 | s=['A', 'G', 'U', 'A', 'U', 'T', 'C', 'G', 'C', 'T']
k=1 | s=['A', 'G', 'U', 'A', 'U', 'T', 'C', 'G', 'C', 'T']
k=2 | s=['C', 'T', 'T', 'A', 'U', 'G', 'G', 'C', 'A', 'U']
k=3 | s=['C', 'T', 'T', 'A', 'U', 'G', 'G', 'A', 'C', 'U']
k=4 | s=['U', 'T', 'A', 'T', 'C', 'G', 'U', 'C', 'A', 'G']
k=5 | s=['U', 'C', 'G', 'T', 'T', 'G', 'U', 'C', 'A', 'A']
k=6 | s=['A', 'C', 'A', 'C', 'U', 'G', 'T', 'T', 'U', 'G']
k=7 | s=['T', 'C', 'A', 'U', 'G', 'U', 'T', 'A', 'C', 'G']
k=8 | s=['A', 'C', 'G', 'T', 'C', 'U', 'U', 'T', 'G', 'A']
k=9 | s=['A', 'U', 'T', 'G', 'C', 'A', 'C', 'G', 'T', 'U']
```

Figure 8.4: A Tabulation Trace of Shuffling the na-Sequence Θ_{palna} from **Equation 10.28** with **Algorithm 3 (FLSA)**.

8.1.3 The Source Code for FLSA

And the essential source code — compartible with **Python 3**, is as provided in **Listing 8.1**

Listing 8.1: The FLSA

```
#!/usr/bin/env python3
import math

def lu_shuffler_a(s, k0):
    n = len(s)
    if n < 2:
        return s,k0 #essentially, nothing to shuffle
    # make k meaningful...
    k = k0 % n
    k1 = k # for tracing/debugging...
    k = 1 if k < 1 else k # can't partition between nothing
    new_s = s
    while True:
        if k == n: #can't partition between nothing
            break
        p_index = k
        p_start = 0
        sub_p_index = math.ceil(0.5 * p_index)
        #print(f"At k={k} |"
        #        +f" [[{p_start}-{sub_p_index}][{sub_p_index}-"
        #        +f"{p_index}][{p_index}-{n}]")
        sub_p_l = new_s[p_start:sub_p_index]
        sub_p_r = new_s[sub_p_index:p_index]
        rest_p = new_s[p_index:n]
        swapped = sub_p_r + sub_p_l # first swap
        if k%2 == 1: # k is odd
            new_s = rest_p + swapped # second swap
            #print(f"At k={k} s -->"
            #+f"[{p_index}-{n}][{sub_p_index}-"
            #+f"{p_index}][{p_start}-{sub_p_index}] == {new_s}")
        else:
            new_s = swapped + rest_p # soft swap
            #print(f"At k={k} s --> [{sub_p_index}-"
            #        +f"{p_index}][{p_start}-"
            #        +f"{sub_p_index}][{p_index}-{n}] == {new_s}")
        k += 1 # so we advance from k-gram to (k+1)-grams
    #print(f"k={k1} | s={new_s}")
    return new_s,k

#---[ Uncomment Lines Below to Run Examples ]
#---[Some Further Experiments]
#s = [1,2,3] # a good base-case scenario..
#s = [0,1,2,3,4,5,6,7,8,9] # the base-10 n-SSI
#s = ['A','C','G','T','U','A','C','G','T','U'] # na-SS+na-SS
#s = ['A','C','G','T','U','U','T','G','C','A'] # na-SS+complement(na-SS)
#---[Generate Lu-Shuffle Look-Up Tables]
#k = len(s)
#s_k = 0
#print(f"GIVEN s={s}")
#while s_k < k:
#    lu_shuffler_a(s,s_k) # for FLSA
#    s_k += 1
```

Figure 8.5: The First Lu-Shuffle Algorithm implemented using Python

Concerning the source code sample for the **FLSA** as shown in **Listing 8.1**,

realize that we have chosen to leave explanatory notes and debugging/tracing code (though commented out) within the sample, mostly to help newcomers learn to read the program well as well as help any nonbelievers or critics of the **FLSA** to try it out themselves and also look at what happens in the guts of the algorithm while it computes the final resultant. This shall help many to also be able to apply or use the algorithm by paper and pen because it is that simple, though unimaginably powerful.

8.2 The Second Lu-Shuffle Algorithm (SLSA)

This second algorithm, is a variation of and enhancement of the **Algorithm 3**. Also, and **IMPORTANT to NOTE**, this second algorithm combines with the first algorithm at each invocation, so that it takes the output of **FLSA** and further transforms it to produce the (***more* randomized**) resultant sequence as we see in the following algorithm specification:

8.2.1 The SLSA Algorithm Using Mathematics

Algorithm 4 (The **Second Lu-Shuffle Algorithm**: $\text{slsa}(\Theta, k_0)$)**.**

1. ***GIVEN*** *source sequence* Θ *of length* $n = \text{ɯ}(\Theta)$.
2. ***GIVEN*** *input parameter* k_0 *that is* ***Initial Partition Index****.*
3. ***IF*** $n < 2$ ***RETURN*** Θ *unmodified.*
4. ***LIMIT*** *initial partition index using cardinality of input sequence, i.e*

$$k = k_0 \mod n \tag{8.3}$$

5. ***IF*** $k < 1$ ***UPDATE****:* $k = 1$.
6. ***INITIALIZE*** *the shuffled sequence,* Θ^* *with the output of invoking* ***Algorithm 3*** *using the given parameters as such:*

$$\Theta^*, k^* = lu\_shuffler\_a(\Theta, n - k) \tag{8.4}$$

IGNORE k^**, while* Θ^* *becomes the* ***initial shuffled resultant sequence*** *for the rest of this algorithm.*

7. ***WHILE TRUE****:*
   (a) ***BREAK IF*** $k = n$
   (b) ***SET*** *partition index,* $p_{index} = k$
   (c) ***SET*** *partition start,* $p_{start} = 0$
   (d) ***SET*** *sub-partition end/index,* $sp_{index} = \textbf{\textit{CEIL}}(\frac{p_{index}}{2})$

(e) ***SET*** *left sub-partition,* $\Theta_l = \{\omega_i | \omega_i \in \Theta^* \quad \wedge \quad i \in [p_{start}, sp_{index}]\}$

(f) ***SET*** *right sub-partition,* $\Theta_r = \{\omega_i | \omega_i \in \Theta^* \quad \wedge \quad i \in [sp_{index}, p_{index}]\}$

(g) ***SET*** *unpartitioned sequence,* $\Theta_{rest} = \{\omega_i | \omega_i \in \Theta^* \quad \wedge \quad i \in [p_{index}, n]\}$

(h) ***SET*** *swapped sequence,* $\Theta_{swap} = \Theta_r \cdot \Theta_l$

(i) ***IF*** k *is* ***ODD****:* $(k \mod 2) = 1$*:*

i. ***UPDATE resultant sequence****,* $\Theta^* = \Theta_{rest} \cdot \Theta_{swap}$

(j) ***ELSE IF*** k *is* ***EVEN****:* $(k \mod 2) = 0$*:*

i. ***UPDATE resultant sequence****,* $\Theta^* = \Theta_{swap} \cdot \Theta_{rest}$

(k) ***INCREMENT*** k*:* $k = k + 1$

8. ***RETURN*** *resultant sequence,* Θ^**.*

□

So, important to note that **Algorithm 4** invokes **Algorithm 3** with the **decreasing initial partitioning index** $n - k$ as shown in **Equation 8.4**

8.2.2 Examples of Applying SLSA

Using the same exact sequences, and in the same order as we utilized in **Section 8.1.2**, we shall come to appreciate readily, not just that the two algorithms, though they perform similar functions, produce **different** results for the same exact sequence and same exact partitioning index parameter — **IMPORTANT:** However, they are both **consistent** — the same input sequence with a particular k shall always produce the expected shuffled sequence post-transform.

Note that **Figure 8.6** corresponds to the earlier table in **Figure 8.1**.

```
GIVEN s=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
k=0 | s=[4, 2, 8, 9, 0, 7, 3, 6, 5, 1]
k=1 | s=[4, 2, 8, 9, 0, 7, 3, 6, 5, 1]
k=2 | s=[7, 5, 3, 0, 8, 2, 6, 1, 9, 4]
k=3 | s=[1, 6, 4, 7, 2, 9, 0, 3, 8, 5]
k=4 | s=[2, 1, 0, 6, 8, 3, 5, 4, 7, 9]
k=5 | s=[7, 1, 2, 4, 6, 8, 3, 9, 0, 5]
k=6 | s=[5, 9, 2, 3, 7, 0, 4, 6, 1, 8]
k=7 | s=[9, 1, 5, 4, 3, 2, 7, 8, 6, 0]
k=8 | s=[5, 6, 3, 9, 0, 4, 2, 7, 1, 8]
k=9 | s=[6, 3, 8, 2, 1, 9, 7, 4, 0, 5]
```

Figure 8.6: A Tabulation Trace of Shuffling the Base-10 n-SSI Sequence with increasing values of k, using **Algorithm 4 (SLSA)**, the **Second Lu-Shuffle Algorithm**.

```
GIVEN s=[1, 2]
k=0 | s=[1, 2]
k=1 | s=[1, 2]


GIVEN s=[1, 2, 3]
k=0 | s=[3, 1, 2]
k=1 | s=[3, 1, 2]
k=2 | s=[2, 3, 1]


GIVEN s=[1, 2, 3, 4]
k=0 | s=[4, 2, 1, 3]
k=1 | s=[4, 2, 1, 3]
k=2 | s=[1, 2, 3, 4]
k=3 | s=[2, 3, 1, 4]
```

Figure 8.7: A Tabulation Trace of Shuffling basic base-case scenarios to help highlight how **SLSA** works and how it relates to **FLSA**.

Note that **Figure 8.8** corresponds to the earlier table in **Figure 8.3**.

```
GIVEN s=['A', 'C', 'G', 'T', 'U', 'A', 'C', 'G', 'T', 'U']
k=0 | s=['U', 'G', 'T', 'U', 'A', 'G', 'T', 'C', 'A', 'C']
k=1 | s=['U', 'G', 'T', 'U', 'A', 'G', 'T', 'C', 'A', 'C']
k=2 | s=['G', 'A', 'T', 'A', 'T', 'G', 'C', 'C', 'U', 'U']
k=3 | s=['C', 'C', 'U', 'G', 'G', 'U', 'A', 'T', 'T', 'A']
k=4 | s=['G', 'C', 'A', 'C', 'T', 'T', 'A', 'U', 'G', 'U']
k=5 | s=['G', 'C', 'G', 'U', 'C', 'T', 'T', 'U', 'A', 'A']
k=6 | s=['A', 'U', 'G', 'T', 'G', 'A', 'U', 'C', 'C', 'T']
k=7 | s=['U', 'C', 'A', 'U', 'T', 'G', 'G', 'T', 'C', 'A']
k=8 | s=['A', 'C', 'T', 'U', 'A', 'U', 'G', 'G', 'C', 'T']
k=9 | s=['C', 'T', 'T', 'G', 'C', 'U', 'G', 'U', 'A', 'A']
```

Figure 8.8: A Tabulation Trace of Shuffling the na-Sequence Θ_{2na} from **Equation 10.27** with **Algorithm 4 (SLSA)**.

Note that **Figure 8.9** corresponds to the earlier table in **Figure 8.4**.

```
GIVEN s=['A', 'C', 'G', 'T', 'U', 'U', 'T', 'G', 'C', 'A']
k=0 | s=['U', 'G', 'C', 'A', 'A', 'G', 'T', 'T', 'U', 'C']
k=1 | s=['U', 'G', 'C', 'A', 'A', 'G', 'T', 'T', 'U', 'C']
k=2 | s=['G', 'U', 'T', 'A', 'C', 'G', 'T', 'C', 'A', 'U']
k=3 | s=['C', 'T', 'U', 'G', 'G', 'A', 'A', 'T', 'C', 'U']
k=4 | s=['G', 'C', 'A', 'T', 'C', 'T', 'U', 'U', 'G', 'A']
k=5 | s=['G', 'C', 'G', 'U', 'T', 'C', 'T', 'A', 'A', 'U']
k=6 | s=['U', 'A', 'G', 'T', 'G', 'A', 'U', 'T', 'C', 'C']
k=7 | s=['A', 'C', 'U', 'U', 'T', 'G', 'G', 'C', 'T', 'A']
k=8 | s=['U', 'T', 'T', 'A', 'A', 'U', 'G', 'G', 'C', 'C']
k=9 | s=['T', 'T', 'C', 'G', 'C', 'A', 'G', 'U', 'A', 'U']
```

Figure 8.9: A Tabulation Trace of Shuffling the na-Sequence Θ_{palna} from **Equation 10.28** with **Algorithm 4 (SLSA)**.

8.2.3 The Source Code for SLSA

And the essential source code — compartible with **Python 3**, is as provided in **Listing 8.2**

Listing 8.2: The SLSA

```
#!/usr/bin/env python3
import math

def lu_shuffler_b(s, k0):
    n = len(s)
    if n < 2:
        return s,k0
    k = k0 % n
    k1 = k
    k = 1 if k < 1 else k
    new_s = lu_shuffler_a(s,n-k)[0] #invoke FLSA with k0=n-k
    while True:
        if k == n:
            break
        p_index = k
        p_start = 0
        sub_p_index = math.ceil(0.5 * p_index)
        sub_p_l = new_s[p_start:sub_p_index]
        sub_p_r = new_s[sub_p_index:p_index]
        rest_p = new_s[p_index:n]
        swapped = sub_p_r + sub_p_l
        if k%2 == 1:
            new_s = rest_p + swapped
        else:
            new_s = swapped + rest_p
        k += 1
    #print(f"k={k1} | s={new_s}")
    return new_s,k
```

Figure 8.10: The Second Lu-Shuffle Algorithm implemented using Python

8.3 Generating Realistic Random na-Sequences

In the previous sections we have considered the matter of taking a particular symbol set or some tiny set of almost or exactly distinct symbols and then worked on ways to generate random sequences from these. However, focus was specifically on randomizing an existing sequence, and not necessarily about generating entirely new sequences with say new symbol distributions different from what was in the source/input sequence. Thus, starting from the fundamental aspect of **generating a SINGLE RANDOM DNA symbol** — such as randomly generating one of the DNA-nucleobases from ψ_{DNA} or doing the same for RNA, shall be where our adventures start.

8.3.1 How to Generate a Random Nucleotide & The Random Symbol Generator

Given that we already have the necessary source symbol sets, the matter for how to generate a single **random letter** from a particular symbol set shall be as follows — first, we shall merely define an abstract machine using the notation and

semantics of transformatics.

Transformer 10 (The **Random Symbol Generator (RSG)**, $RSG(k, \Theta_n)$)**.**
$\Theta_n \xrightarrow{O_{RSG}(k,\Theta_n)} \Theta_k^*$;
$\text{ш}(\Theta_k^*) = k \quad \wedge \quad \psi(\Theta_k^*) \subseteq \psi(\Theta)$
$\wedge \quad \forall x, y \in \mathbb{N}, \Omega : N \times \psi_y,$

$$RSG(x, \Omega) = SELECT(x, SHUFFLE(PROTRACT(\psi_y, k \times 2 \times \text{ш}(\psi_y))))$$

Where;

- $\psi_y = USYMBOLSET(\Omega)$
  but in a more robust case, we can also randomize the symbol set to be used in generating the random sequence, by generating ψ_y *thus:*

$$\psi_y = SHUFFLE(\, USYMBOLSET(\Omega)) \tag{8.5}$$

- ***USYMBOLSET***(Ω) *is a function that reduces the input argument* Ω*, which is expected to be a sequence of some symbols — unique or not, to its* ***unspecific symbol set*** — $\psi(\Omega)$[a].
- ***PROTRACT***(Θ, n) *is a transformation of the sequence of [possibly] distinct* n *symbols from the source sequence* Θ *such that the resultant sequence is of length*[b] $k \times 2 \times \text{ш}(\psi_y)$ *and spans the source sequence symbol set*[c].
- ***SHUFFLE***(Θ) *is a function such as a generalization of* ***lu_shuffler_b(·)*** *from* ***Algorithm 4*** *that returns a shuffled/randomized instance of any input sequence such as* Θ *that it is given.*
- ***SELECT***(x, Θ) *is a function that can return the first* x *terms from* Θ *as the result of its invocation for some* $x \leq \text{ш}(\Theta)$.

□

[a] See **Definition 4** in [21]

[b] But just like the first parameter might be any sequence of symbols, also, the second parameter could be any positive number, and by which, the function shall **either shrink/truncate or grow/multiply** the input sequence to the required **multiplier** size

[c] As you shall see in the sample outputs and as well as the actual implementation source code for our **RSG**, the choice of the protraction parameter $k \times 2 \times \text{ш}(\psi_y)$, is both to be sensitive to the length of the input sequence's unique symbol set, but also the size of final output required — both factors influence the quality of the result in terms of randomness/entropy.

So, with the essential mathematics out of the way as depicted in **Transformer 10**, we can then try to see how this would work practically with some real tangible

computer programs that implement that generator-transformer specification.

8.3.2 The USYMBOLSET Function

We shall go bit by bit. So first, here is what the **USYMBOLSET** function looks like in Python:

Listing 8.3: The USYMBOLSET

```
#!/usr/bin/env python3

#---[ USYMBOLSET(s) ]
def usymbolset(s):
    n_s = len(s)
    if n_s <= 1:
        return s

    # the next commented-out code would kinda work,
    # if we just wanted a dirty and pythonic solution, but
    # it doesn't respect order of first occurrence!
    #resultant = list(set(s))

    #our CORRECT solution for unspecific symbol set..
    resultant = []
    for i in range(n_s):
        s_i = s[i]
        if not (s_i in resultant):
            resultant.append(s_i)
    return resultant

#---[ TEST USYMBOLSET ]
#s = [0,1,2,3,4,5,6,7,8,9]
# s = [0,1,2,3,4,5,6,7,8,9] --> usymbolset(s+s+s+s)
#--> [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
#s = [1,0,1,1,1,1,1,0,1,1] --> usymbolset(s) --> [0,1]
#s = "3.1415926535898" # --> usymbolset(s) -->
#['3', '.', '1', '4', '5', '9', '2', '6', '8']
#s = ['A' , 'C' , 'G' , 'T' , 'U' , 'U' , 'T' , 'G' , 'C' , 'A'] #w/duplication
# usymbolset(s) --> ['A', 'C', 'G', 'T', 'U']
#result = usymbolset(s)
#print(result)
```

Figure 8.11: The **USYMBOLSET**(Ω) unspecific symbol set generator implemented using Python

8.3.3 The PROTRACT Function

We shall go bit by bit. So first, here is what the **PROTRACT** function looks like in Python:

Listing 8.4: The PROTRACT

```
#!/usr/bin/env python3
import math

#---[ PROTRACT(s,n) ]
def protract(s,n):
    n_s = len(s)
    if n_s < 1:
        return s
    if n < n_s:
        return s[:n]
    if n == n_s:
        return s
    multiple = math.ceil(n_s / n)
    target_n = n * multiple
    resultant = s
    i = 1
    while i < target_n:
        resultant = resultant + s
        i += 1
    return resultant[:n]

#---[ TEST PROTRACT ]
# s = [0,1,2,3,4,5,6,7,8,9], n = 3 --> [0,1,2]
# s = [0,1,2,3,4,5,6,7,8,9], n = 23 -->
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2]
#result = protract(s,23)
#print(result)
```

Figure 8.12: The **PROTRACT**$(\psi_y, 2x)$ sequence generator-transformer implemented using Python

8.3.4 The Random Number Generator, RNG Function

Before we proceed to the **SHUFFLE** function, we shall want to first specify and test an important utility that would otherwise make our implementation dependent on someone else's random sequence generator — thus rendering all our efforts here almost useless!

The critically important function is **RNG**(ll, ul) that essentially takes two parameters — ll, the **lower limit** and ul, the **upper limit**, both of which are expected to be integers, and using which, the function is supposed to generate a **random number** n that lies within the range:

$$ll \leq n \leq ul \tag{8.6}$$

It is such a CRITICALLY important routine(see author's earlier work on the matter[34]), however, for the interests of keeping things simple, for now, we shall settle with a basic implementation that only relies on the **System Time** functions as our **source of entropy**[4].

[4]While, and perhaps in the near future, we continue to seek for theory and practical ways to generate entropy algorithmically — which might be *futile*, or perhaps implement our own entropy generators interfacing with entropy sources in nature?

So, without delving into formally specifying our **Random Number Generator**, **RNG** here, the associated definition of this utility as implemented using Python is as follows:

Listing 8.5: The RNG

```
#!/usr/bin/env python3
import time

#---[ RNG(ll,ul) ]
def random_ng(l,u):
    lu = max(l,u)
    ll = min(l,u)
    lu = lu + 1 if ll == lu else lu
    part_k = 1
    candidates = lu_shuffler_a(list(range(ll,lu+1)),part_k)[0]
    n = len(candidates)
    pick = int(time.time())%n #because we need an entropy source!
    return candidates[pick]

#---[ TEST RNG ]
#for r in [[0,10],[0,1],[0,100],[1900,1999]]:
#    print(f"RNG From {r[0]}-{r[1]} -> {random_ng(r[0],r[1])}")
```

Figure 8.13: The **RNG**(ll, ul) random number generator implemented using Python

We can indeed look at some trace dumps from testing our beautiful RNG and appreciate what a little gem it is!

Listing 8.6: A TESTING Our RANDOM NUMBER GENERATOR

```
RNG From 0-10 -> 6
RNG From 0-1 -> 0
RNG From 0-100 -> 56
RNG From 1900-1999 -> 1984

RNG From 0-10 -> 7
RNG From 0-1 -> 1
RNG From 0-100 -> 46
RNG From 1900-1999 -> 1950


k=1 | s=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
RNG From 0-10 -> 2

k=1 | s=[0, 1]
RNG From 0-1 -> 1

k=1 | s=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
    20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37,
   38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56,
    57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74,
   75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93,
    94, 95, 96, 97, 98, 99, 100]
RNG From 0-100 -> 28

k=1 | s=[1900, 1901, 1902, 1903, 1904, 1905, 1906, 1907, 1908, 1909, 1910,
   1911, 1912, 1913, 1914, 1915, 1916, 1917, 1918, 1919, 1920, 1921, 1922,
   1923, 1924, 1925, 1926, 1927, 1928, 1929, 1930, 1931, 1932, 1933, 1934,
   1935, 1936, 1937, 1938, 1939, 1940, 1941, 1942, 1943, 1944, 1945, 1946,
   1947, 1948, 1949, 1950, 1951, 1952, 1953, 1954, 1955, 1956, 1957, 1958,
   1959, 1960, 1961, 1962, 1963, 1964, 1965, 1966, 1967, 1968, 1969, 1970,
   1971, 1972, 1973, 1974, 1975, 1976, 1977, 1978, 1979, 1980, 1981, 1982,
   1983, 1984, 1985, 1986, 1987, 1988, 1989, 1990, 1991, 1992, 1993, 1994,
   1995, 1996, 1997, 1998, 1999]
RNG From 1900-1999 -> 1987
```

Figure 8.14: A text dump from testing our RNG(ll, ul) that only utilizes System Time as entropy source, and uses our First Lu-Shuffle Algorithm to generate True Random Numbers

Before we proceed, just note that the verbose output in the lower trace dump in **Listing 8.6** is because we turned on the DEBUG print in our **lu_shuffler_a**($\cdot$) invocation — see **Line#11**, so as to cross-check/confirm if our RNG is operating on the correct ranges given our input lower and upper bounds.

8.3.5 The SHUFFLE Function

And here is what the **SHUFFLE** function looks like in Python:

Listing 8.7: The SHUFFLE

```
1  #!/usr/bin/env python3
2
3  #---[ SHUFFLE(s) ]
4  def shuffle(s):
5      n_s = len(s)
6      if n_s < 2:
7          return s
8      random_k = random_ng(1,n_s)
9      shuffled_s,k = lu_shuffler_b(s,random_k)
10     random_k2 = random_ng(1,n_s)
11     shuffled_s,k = lu_shuffler_a(shuffled_s,random_k2)
12     return shuffled_s
13
14 #---[ TEST SHUFFLE ]
15 #s = [0,1,2,3,4,5,6,7,8,9]
16 # s = [0,1,2,3,4,5,6,7,8,9] --> [9, 3, 4, 7, 0, 5, 2, 1, 6, 8],
17 # [4, 6, 0, 3, 5, 2, 7, 9, 1, 8], etc.
18 #s = ['A' , 'C' , 'G' , 'T' , 'U']
19 # s = ['A' , 'C' , 'G' , 'T' , 'U'] --> ['C', 'G', 'T', 'U', 'A'],
20 # ['U', 'G', 'T', 'C', 'A'], etc.
21 #result = shuffle(s)
22 #print(result)
```

Figure 8.15: The **SHUFFLE**(Θ) sequence randomizer implemented using Python

Definitely, just like we didn't want to implement our own basic random number generator (see **Line#10** and **Line#12** in **Listing 8.7**), we could also have used an external shuffle algorithm — such as with the standard python random utility in `s=[1,2,3];rng.shuffle(s);s` that might return `[3,1,2]` for example, but we chose to use our own shuffle algorithm as developed in the previous sections (see **Line#11** and **Line#13** that invoke the **SLSA**, **lu_shuffler_b(·)**, defined in **Listing 8.2**), so those who need to cut dependencies might pick a leaf from our solution, or perhaps port our algorithms to any programming language. Also, unlike invoking **lu_shuffler_b**(Θ, k) directly, the provided sequence randomizer utility function takes care of generating random parameters to the **Lu Shuffle Algorithm**, by randomly picking legitimate values of k from 1 to $\text{ш}(\Theta)$, so that the **SHUFFLE** function can be invoked without having to explicitly pass any arguments other than just the sequence one wishes to have shuffled.

Also, note that **Listing 8.7**, like the ones we have looked at before, provides some (commented out) code that might give one an idea how to use the function, or how the potential results of invoking it with certain sequences might look like.

8.3.6 The SELECT Function

And as for the **SELECT** function required by **Transformer 10** (the **RSG**), the corresponding definition for that function in Python is as shown:

Listing 8.8: The SELECT

```
#!/usr/bin/env python3

#---[ SELECT(n,s) ]
def select(n,s):
    n_s = len(s)
    resultant = []
    if n == 0:
        return resultant
    n = n if n <= n_s else n_s
    resultant = s[:n]
    return resultant

#---[ TEST SELECT ]
#s = [0,1,2,3,4,5,6,7,8,9]
#s = [0,1,2,3,4,5,6,7,8,9] --> select(2,s) --> [0,1],
# select(12,s) --> [0, 1, 2, 3, 4, 5, 6, 7, 8, 9], etc.
#s = ['A' , 'C' , 'G' , 'T' , 'U']
# s = ['A' , 'C' , 'G' , 'T' , 'U'] --> select(3,s) --> ['A', 'C', 'G'], etc.
#result = select(12,s)
#print(result)
```

Figure 8.16: The **SELECT**(n, Θ) sequence sampler function implemented using Python

8.3.7 The RSG Function

Having seen how to implement the individual essential components that make the **Random Symbol Generator (RSG)**, $RSG(k, \Theta_n)$ defined in **Transformer 10**, we can then go ahead and actually implement a re-usable, and genetic random sequence or even random symbol generator. As with the sample codes offered in this section, we shall merely default to implementing a Python-compliant function, but the semantics and code could of course be adapted for other programming languages by interested engineers, students and researchers.

Listing 8.9: The RSG

```
#!/usr/bin/env python3

# import/add earlier utilities as necessary...
# usymbolset(s), protract(s,n), shuffle(s) and select(n,s)

#---[ RSG(n,s) ]
def rsg(n,s):
    n_s = len(s)
    resultant = []
    if n == 0 or n_s == 0:
        return resultant
    # first, compute the u-symbol set
    s_usymbolset = usymbolset(s)
    n_us = len(s_usymbolset)
    # then chain the transformers...
    resultant = shuffle(s_usymbolset) # randomize the symbol set
    resultant = protract(resultant,n*2*n_us)
    resultant = shuffle(resultant)
    # then pick only as much as was asked for...
    resultant = select(n,resultant)
    return resultant

#---[ TEST RSG ]
#s = [0,1,2,3,4,5,6,7,8,9]
#s = [0,1,2,3,4,5,6,7,8,9] --> rsg(0,s) --> [],
# rsg(1,s) --> [6], [1], [0], etc.
# rsg(3,s) --> [2, 9, 5], [0, 1, 2],[5, 6, 5], etc.
#s = ['A' , 'C' , 'G' , 'T' , 'U']
# rsg(1,s) --> ['T'], ['A'], ['G'], etc.
#s = ['A' , 'T' , 'C' , 'G']
# rsg(3,s) --> ['C', 'G', 'A'], ['G', 'T', 'C'] # random DNA codons!
#result = rsg(3,s)
#print(result)
# what of generating sentences from words?
# e.g pass in a glossary
# rsg(10,['CAT','DOG','PEN','ACE','SKY','EYE','IS','THAT','NO','YOU'])
# --> ['YOU', 'IS', 'NO', 'DOG', 'SKY', 'THAT', 'ACE', 'EYE', 'CAT', 'PEN']
```

Figure 8.17: The **RSG**(n, Θ) random sequence generator function implemented using Python

Of course, as shown in the final comments in the **RSG** code listing in **Listing 8.9**, we can use that basic function to **generate random ANYTHING!** It could be used to generate random numbers — whether individual digits such as 1, 5 or 0,.. but also sequences or n-grams of them — just depending on what kind of sequence we pass in as the source, how long we wish the output to be and what we do with the result.

The **RSG** could be used to compute long DNA sequences using a clever invocation such as:

```
 N=3*200; print(''.join(str(c) for c in
rsg(N,['A','T','C','G'])));
```

As shown in the following listing, does indeed perform impressively!

Listing 8.10: A RANDOM DNA SEQUENCE

```
TATACGGCAAGCGATGAGATATCAAGCTAATCGATAGGCAACGAGACACCAGGAGACGCGT
CAGGCAACAAGGAATTAAGTCACGCGTCAAGCTATGAGATACTAAGCAACACGGTAGGCAA
CGAGACAATAGGAGATGCGCCAGGCGACAAGGAATCAAGTTATCCGACAAGCAATGAGATA
GTAAGCGACGCGGTAGGCAACGAGACATAAGGAAACGCGCCAGGCCACAAGGAATCAAGTT
ATACGGCAAGCGATGAGATATCAAGCTATTCGATAGGCAACGAGACACCAGGAGTCGCTTC
CGTCAACATCGAACTCATTCCCGCCTCAACCTATGCCATACTAACCAATACCGTTGCCAAC
GATACAACTGCAGTTGCTCCCGTAGACATCGATCCCATTTCTCACACAACCATTGCCATAG
TAACCGAGGACGTTGCCAGCGATACATTTGCAATTGCTCCCGTACGCATCGCTCACATCTT
GAGTGCTATAGTCGCTACGTTCATCTTCTGTACTGTCAACGCTACGCCCGTCGTGGCTTCT
GTGACCACTGCTTACATCCTCGATTCTATGTTCGCTACACGCATCATAACT
```

Figure 8.18: A Random DNA Sequence of exactly 200 random codons as generated using our **RSG**(n, Θ) random sequence generator algorithm.

The sheer significance of having a properly working generator-transformer such as the **RSG** is difficult — almost impossible, to explain.

Chapter 9

Gene Expression in Living Organisms Leveraging Genetic Code (DNA $\rightarrow mRNA \rightarrow Protein \rightarrow$ Organism)

> `Code: A rule for transforming a message from one symbolic form (the source alphabet) into another (the target alphabet), usually without loss of information. The process of transformation is called encoding and its converse is called decoding.`
>
> — **The Oxford Companion to the Mind**[12]

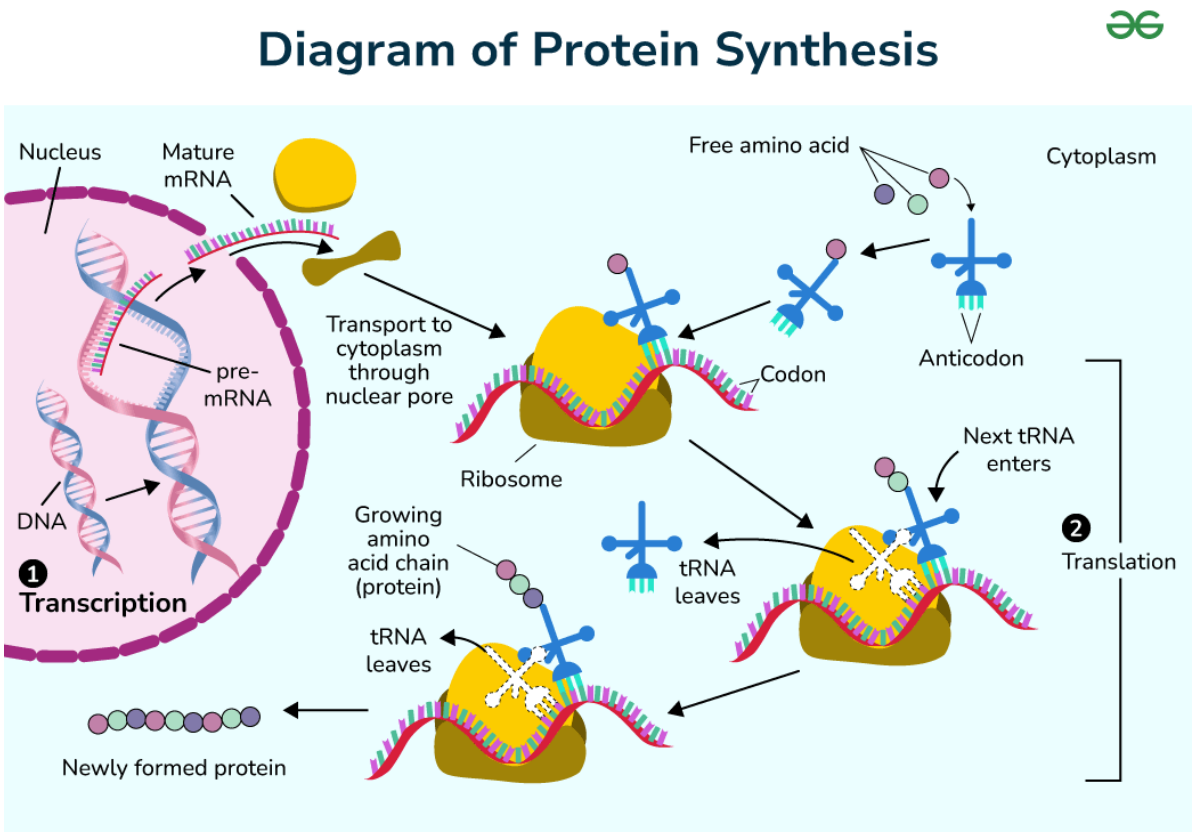


Figure 9.1: An illustration of the gene translation process in a cell via protein-factories known as ribosomes[2]

In later section (**Chapter 10**) we shall deal with [whole] genome expression,

and shall especially deal with hypothetical systems and use the modal sequence concept as the basis for exploring expression and/or manifestation of an entire organism from just its basic genetic code. In this section though, we shall focus on actual expression in real and natural biological systems, and shall focus more on expression at a molecular level — essentially, at the level of protein manifestation via **gene expression**.

Definition 10 (The **DNA and RNA Codon Symbol Sets**, ψ^*_{DNA} and ψ^*_{RNA})**.** *If Θ_n is a DNA sequence of length n — meaning, it contains n nucleotides or n symbols from ψ_{DNA}, then, if each triplet of symbols in Θ_n is replaced by its corresponding 3-gram/CODON, then, we can re-write Θ_n as a sequence of terms from a special symbol set ψ^*_{DNA}:*

$$\psi^*_{DNA} = \langle \alpha_1, \alpha_2, \cdots, \alpha_{64} \rangle \tag{9.1}$$

Such that:

$$\mathbb{u}(\psi^*_{DNA}) = \mathbb{u}(\psi_{DNA})^3 = 4^3 = 64 \tag{9.2}$$

*And that each distinct codon has a corresponding standard name associated with either its corresponding amino-acid or whether it is a STOP-codon[12][35]. We shall refer to ψ^*_{DNA} as the **DNA Codon Symbol Set**.*
*Further, since RNA sequences are similar to DNA sequences with just the symbols 'T' swapped with 'U' (see **Definition 4** and **Transformer 11**), we can also expect that for RNA sequences, there would be an equivalent symbol set for RNA-codons as such:*

$$\psi^*_{RNA} = \langle \beta_1, \beta_2, \cdots, \beta_{64} \rangle \tag{9.3}$$

And the names of the codons in ψ^*_{RNA} are similar to those in ψ^*_{DNA} by the following rule:

$$name(\beta_i) = name(\alpha_i) \quad \forall \beta_i \in \psi^*_{RNA} : \beta_i = O_{mRNA-encode}(\alpha_i) \tag{9.4}$$

It shall be interesting to note that for genetic code in natural automata (nature-like bio-automata[1] and generally living things), the most important reason the genetic coding language exists, is so that the body/host-organism system can produce required materials as and when they are needed or demanded for. This for example means producing new or extra body tissue in a still growing organism or in one with any damaged or missing tissue, and essentially, such productions are about the synthesis of particular molecules in the body's system that are basically proteins. The diagram in **Figure 9.1** is a basic illustration of the process for living organisms — eukaryotes especially.

For simplicity's sake, we can assume the following summarization of the basic process that fully and correctly breaks down the typical ordeal:

First, we shall assume that given the protein is just a chain of amino-acids,

[1]My research assistant — thanks **Microsoft Copilot**, did bring it to my attention that the term "bio-automata" is "usually reserved for models or conceptual representations of biological systems — especially those designed to simulate behaviors, growth patterns, or decisions-making processes using predefined rules, like in cellular automata or agent-based modeling." And thus, much as I often find it attractive to use the term — as an umbrella term including actual living organisms which, from the perspective of the computer scientist in me, are still correctly classifiable under the "biological automata" category in my opinion since they actually operate on some infallible inherent natural program in their DNA. But, I shall adhere to the advise of my assistant for now.

we might as well just think of it as though it were an ordered sequence of some terms, and thus, in keeping with the notation from transformatics, we might just refer to such a protein with our usual typical **resultant sequence** symbol — Θ^*.

And so, given that these proteins are actually nearly direct/1-to-1 mappings from the corresponding DNA sub-sequence code of a finite length, we might then refer to the DNA sequence that encodes the instructions for producing Θ^* with just the basic typical transformatics **source sequence** symbol: Θ — more conveniently, because we wish to also talk of the length of the sequence, we might preferably write the DNA sequence code of length **n** (meaning for example, **it contains exactly n DNA-codons**), as Θ_n. So, for example we might more fully express Θ_n as such:

$$\Theta_n = \langle \alpha_{ij}, \rangle; \alpha_{ij} \in \psi^*_{DNA} \quad \forall j \in [1, n], i \in [1, 64] \quad \wedge \quad n \in \mathbb{N} \tag{9.5}$$

Equation 9.5 being just a sometimes preferable way to write the same exact sequence as:

$$\Theta_n = \langle a_1, a_2, a_3, ..., a_i, ..., a_{n-1}, a_n \rangle; \forall i \quad \exists a_i \in \psi^*_{DNA} \quad \wedge \quad n \in \mathbb{N} \tag{9.6}$$

We have defined the special symbol sets ψ^*_{DNA} in **Definition 10** and ψ_{DNA} in **Definition 1**, and as for ψ^*_{DNA}, we of course know that it essentially is the set of the distinct 64 codons (see **Table 1** in [12]) that *especially* encode amino acids, and which were first introduced in **Chapter 1**. Another way to expound on this is by saying that:

$$\Theta_n = \{a_i \quad | \quad a_i = \prod_{\rho \in \psi_{DNA}}^{3} \rho \quad \wedge \quad \forall i \in [1, n], n \in \mathbb{N} \quad \wedge \quad \psi_{DNA} = \langle A, C, G, T \rangle\} \tag{9.7}$$

Thus we might encounter a gene such as Θ_4 composed of exactly 4 codons as shown below:

$$\Theta_4 = \langle \langle A, T, G \rangle, \langle A, A, A \rangle, \langle T, T, A \rangle, \langle T, A, G \rangle \rangle \tag{9.8}$$

Which, might also equivalently be expressed as a flattened sequence if the fact that the nucleobases it contains are always read in triplets/3-grams/tuples of 3 at a time. So that we can merely write it as:

$$\Theta_{4\times 3} = \langle A, T, G, A, A, A, T, T, A, T, A, G\rangle \tag{9.9}$$

So we imply that under the flat-structure notation, the sequence has exactly 4×3 elements. By this fact and the observation that the previous nested notation merely helps to group together each codon's members within a meaningful sub-sequence, and that the order is otherwise maintained in the flat-sequence structure, we might then also equivalently express the same actual DNA code sequence as an $n \times 3$ matrix as shown below:

$$\Theta_{4\times 3} = \begin{pmatrix} A & T & G \\ A & A & A \\ T & T & A \\ T & A & G \end{pmatrix} \tag{9.10}$$

Whether to actually express it as a $3 \times n$ matrix or as $n \times 3$ might be up to the particular taste of the mathematician or scientist, but otherwise, we know that the ordered sequence Θ in any of the forms above is essentially a **genetic program** to guide a DNA code processor such as a ribosome construct a corresponding protein based on the equivalent transcribed mRNA code sequence Θ_4^* written in mRNA code as such:

$$\Theta_4^* = \langle\langle A, U, G\rangle, \langle A, A, A\rangle, \langle U, U, A\rangle, \langle U, A, G\rangle\rangle \tag{9.11}$$

Which is what we would obtain after a necessary **DNA** $\rightarrow$ **mRNA** transform attainable via a DNA sequence transformer we might define as such:

Transformer 11 (DNA to mRNA Encoder). $\Theta_n \xrightarrow{O_{mRNA-encode}(\cdot)} \Theta_n^*$;
$\forall a_i \in \Theta_n = \langle a_i, \rangle : n, \quad a_i \in \psi_{DNA} \equiv \{A, T, C, G\}$,
and if $\exists a_i \in \Theta_n : a_i = T \implies \exists a_i^* \in \Theta_n^* : a_i^* = U$
Otherwise $a_i = a_i^* \implies \mathit{ш}(a_i = T \in \Theta_n) = \mathit{ш}(U \in \Theta_n^*) \quad \wedge \quad \mathit{ш}(\Theta_n) = \mathit{ш}(\Theta_n^*) = n.$

Thus, though the gene in its DNA form comprised of the ordered sequence of amino-acid codes named as in **Table 9.1**, and yet, the resultant sequence after applying **Transformer 11** to Θ_4 would be as explained in **Table 9.2**.

i	a_i	**Amino Acid (Code-name)**	**Function**
1	ATG	Methionine (Met)	Start Codon: initiates translation
2	AAA	Lysine (Lys)	Basic amino acid
3	TTA	Leucine (Leu)	Non polar amino acid
4	TAG	Stop (Amber)	Terminates translation

Table 9.1: Amino-Acid Codes and Names in Θ_4, a DNA-encoded gene

i	a_i	**Amino Acid (Code-name)**	**Function**
1	AUG	Methionine (Met)	Start Codon: initiates translation
2	AAA	Lysine (Lys)	Basic amino acid
3	UUA	Leucine (Leu)	Non polar amino acid
4	UAG	Stop (Amber)	Terminates translation

Table 9.2: Amino-Acid Codes and Names in Θ_4^*, a mRNA encoded gene

So, note that the names (and code-names) of the mRNA encoded codons stay the same as those of their corresponding DNA-encoded codons in both tables — this is actually generally/conventionally so. But also, note that the functions of the individual codons in either scenario are likewise expressed the same. So, this is is because, when the genetic code is actually being executed (such as in standard protein-synthesis), the processor (the ribosome) merely operates on the mRNA-encoded gene and not directly on the original DNA-encoded code sequences.

Also, important to note, the processor only produces an amino acid (as part of the protein synthesis program), only after having encountered a "start" instruction, and we know that such instructions are the kind encoded by **start codons**, of which the most universally utilized START-codon is **ATG/AUG** known as Methionine, but also other rare-scenario[2] START-codons include the mRNA codes GUG and UUG — used as such in prokaryotes, and then AUU and AUA, used as such in humans only.

[2]They are used less frequently, mostly in prokaryotes and some organelles. When used as START codons, they still recruit the initiator tRNA and translate as **methionine**, not their usual amino acid[24]

And then, the processor will stop the protein construction task once it encounters a gene instruction of the "stop" kind. These are encoded using the **stop codons**, and these are strictly any one of: TAA/UAA (Ochre), TAG/UAG (Amber) and TGA/UGA (Opal)[42].

That said, further note that, after processing the gene, and/or after encountering a stop-codon, the ribosome (also understood as the "protein factory") is then triggered to detach (from the "assembly line") and then release/return the final assembled protein thus far. These resultant proteins are basically just a chain of **actual amino acids** generally starting with the Met-amino acid.

And then, further note that, in case any codons were encountered before the AUG (or rather *start-codon*), these shall then be merely be skipped — they aren't processed or won't translate into any product such as the usual case of producing an amino-acid (this, even if they would normally have triggered the production of some amino acid).

9.1 Protein Manufacturing Algorithm

So, overall, we might sum up this critically important protein generation process with a convenient formalism such as with a protein-production algorithm expressed as in the flow-chart depicted in **Figure 9.2**.

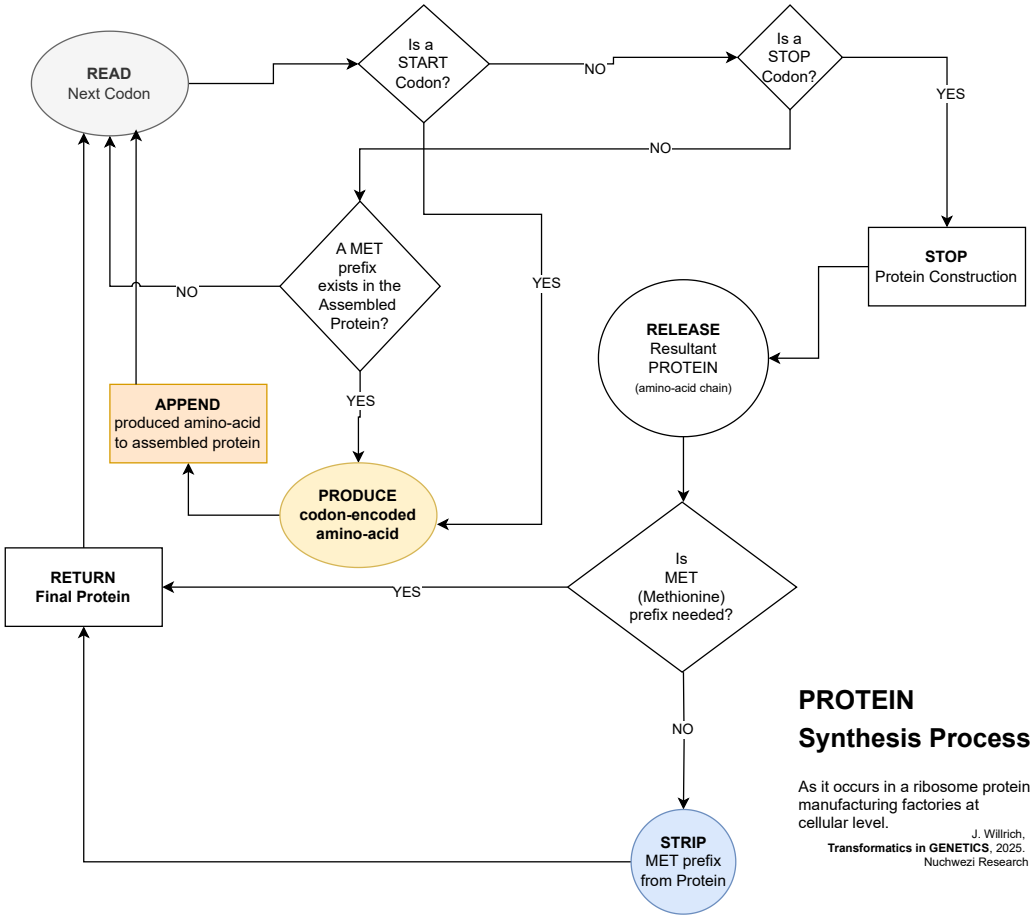


Figure 9.2: **Flow-Chart summarizing the Ribosome-based Protein-Synthesis Process in a Living Cell**

That process which is depicted in **Figure 9.2** is how the ribosome protein-manufacturing factory operates at a cellular level as depicted using a **Flow Chart Diagram** — meaning, the states of the operating environment as well as those of the operator (the ribosome) are interlaced with decision-making scenarios so as to bring to mind the logic behind how the process proceeds. However, in a different diagram — the **Ribosome State Machine** as depicted in **Figure 9.3**, we clearly abstract everything else away and focus on what actually happens from the point-of-view of the gene code sequence processor — the ribosome. In a way, that state machine not only depicts the various states the ribosome shall be in while operating on incoming gene-code sequences (kind of *requests for solutions/solution-instances/proteins* to problems/specifications/genes) and then how it goes about producing the out-going amino-acid sequences (the proteins). It might even start to feel like the ribosome is a kind of 3D-printer, which, when presented with the specifications of a particular 3D-sequence, knows to process it (translate it) and then produce the required/specified object that is in the context

of biological systems we are looking at here, essentially proteins[3].

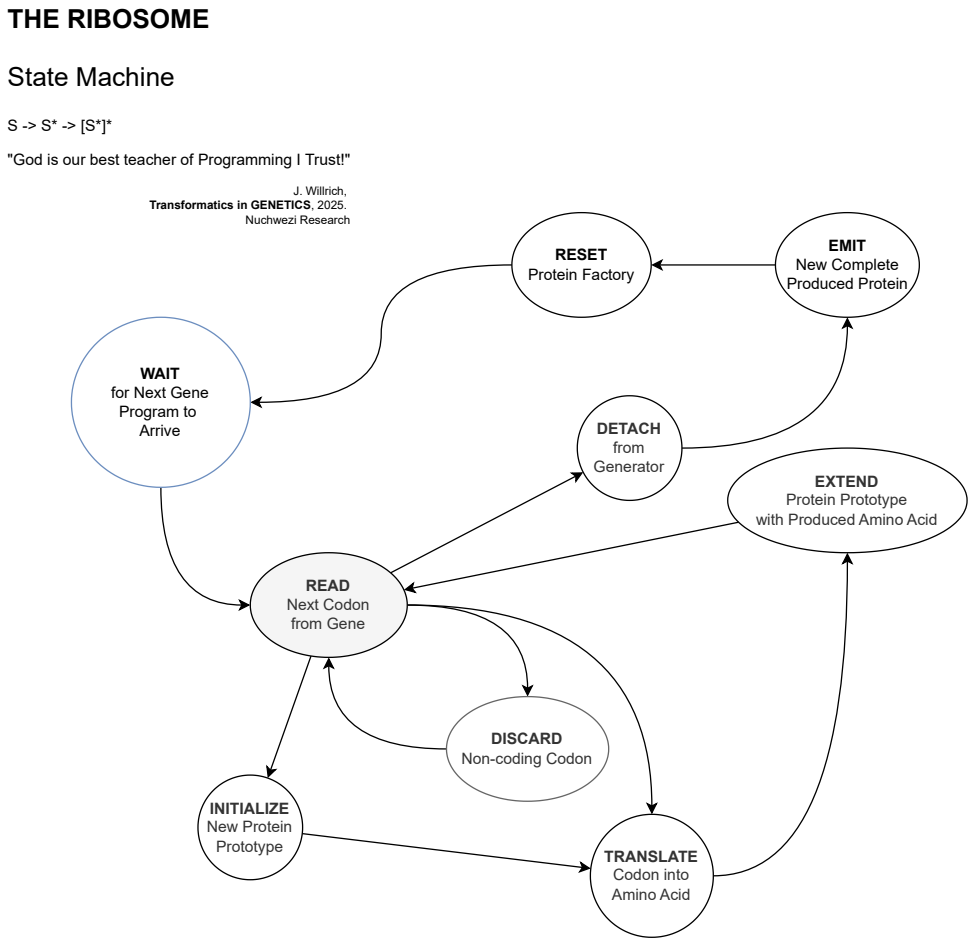


Figure 9.3: **The RIBOSOME State Machine**

Before concluding this section on biological computing systems of the sequence generator kind[44], it should be worthwhile noting that, as depicted in **Figure 9.3**, the process that is naturally found in every living thing's tiniest cells (excluding viruses as already saw in **Chapter 1**, could be, as with any legitimate state machine, be used to implement a proof-of-concept ribosome processor (or a *ribosome computer* — a way to implement bio-automatons that behave like a ribosome or which process code similar to DNA and mRNA, or which generally operate on sequences to produce other (possibly more complex) sequences. Thinking of a robotic ribosome might not be something that is immediately required in contemporary medicine or in-organism computing systems, but might be a concept worth adapting or exploring for the design and implementation of self-contained robotic factories for manufacturing and outputting complex products producible via use of a sequence-processing, assembly-line kind of method just as proteins

[3]Of course, for someone with a background in computer science and who also has interest in designing not just computer programs but also new kinds of computers - abstract machines or not, studying the ribosome from a computing theory and computer architecture perspective — such as so one appreciates what Instruction Set the ribosome employs; how the ribosome compares to say a Von-Neumann architecture machine; is it Turing Complete or not? How might a pure-text processing languages such as TEA[43] implement a ribosome simulation in say a web-browser environment?[38] or that failing, at least allow for the creation of a protein generator or even an entire organism generator as a simulation of how gene code sequences can be translated into sequences of multi-dimension objects?[44] etc.

are produced in a bio-cell by the ribosome. We might for example think of "a sequence from which a finished particular type of car can be manufactured at will" or "a sequence from which certain kinds of sequence-form/sequence-based[4] artificial and/or organic creatures or bots might be systematically produced at will or on-demand". These kinds of printers — which, unlike just printers of things on paper, or 3D-printers that know to only produce plastic variants of models they are fed with, but which can say *print a human*[5], teleport a cow or a banana, etc. might be interesting to explore, as we look into the far-future, where, with humanity's ability to travel and survive in remote and/or unnatural worlds away from their biological home environments (such as Earth's biosphere), might compel them to have to develop new kinds of machines that would not only print out ideas on paper, but also complete food ready-to-eat, medicines to particular kinds of ailments, certain kinds of companion creatures or species, etc. Means to survive on/in alien worlds by leveraging smart, general sequence processors and generators. Talking of which, the next section shall help us start to appreciate this perspective of using the case of genetic code sequences and the ribosome sequence processor into the dimension of both artificial as well as conceptual or hypothetical bio-machines that like the bio-cell can produce complex things via processing of some kinds of code sequences. Bio-automata.

9.2 Gene Expression in Living Things and for Bio-Automata via Ribosomes

One might begin by wondering: **using the concepts from transformatics and pure mathematics, how might we model or express a general and realistic ribosome?**

A plausible and meaningful solution to this problem would be to begin by acquiring or developing *a rigorous and correct working definition of what a ribosome is.* The answer would follow directly from that, thus our first definition in this section; a formal definition of a ribosome in any system natural or artificial:

Definition 11 (A **Ribosome**)**.** *Assuming we re-write a sequence of DNA code in terms of the ordered sequence of nucleotides it contains, as in* ***Equation 9.7*** *re-written as in* ***Equation 9.12****:*

[4]Vertebrates anyone(?)

[5]Though we might touch on it in a future paper on philosophy — e.g in *Computational Mysticism*[45], it might be interesting to air-out the author's illuminating view that unlike most other bio-mata such as beasts in the wild or fish in the seas, and definitely not as with silicon-based automatons such as GPT-powered modern *disincarnate* artificial entities — nor the likes of the ZHA qAGI[46], that the human in particular, has this peculiar attribute to them that, apart from just their material substratum as any physical robot might possess or need — and which is say the domain of "material producers" such as the ribosome is, and away from their conceptual/software substratum too, they seem to, or perhaps arguably, also possess a preternatural layer of existence that perhaps is or might not have anything to do with their DNA/material-blueprint. A better or more precise classification term for such [*preter*-]intelligent -mata/matter(?) might be the still underground term and concept of a *Psymaton* or **Psymata** — bits of this line of discourse have been already touched on in the Psymaz Interview[47].

$$\Theta_n = \{a_i \quad | \quad a_i = \prod_{\rho \in \psi_{DNA}}^{3} \rho\} : n \tag{9.12}$$

*Θ_n would then be any sequence of DNA-codons of length n, equivalent to an equivalent flat-structure ordered sequence of nucleotides of length $n \times 3$. We can then produce mRNA-codons from Θ_n as per **Transformer 11**, so that we produce a new mRNA-codon sequence of length n that is generated as such:*

Transformation 17. $\Theta_n \xrightarrow{O_{mRNA-encode}(\cdot)} \Theta_n^*$;

$\Theta_n^* = \{a_i^* \quad | \quad a_i^* = \prod_{\rho \in \psi_{mRNA}}^{3} \rho\} : n$

And with Θ_n^* produced, we can then merely generate the corresponding ordered sequence of amino-acids, denoted as $[\Theta_n^*]^*$, via the following mRNA to amino-acid transformer:

Transformer 12 (mRNA to Amino-Acid Translator). $\Theta_n^* \xrightarrow{O_{mRNA-translate}(\cdot)} [\Theta_n^*]^*$;

1. *$\mathfrak{u}([\Theta_n^*]^*) < \mathfrak{u}(\Theta_n^*)$ because we only count each codon in the source once, and as per the rules of gene processing/transcription, all non-coding codons (introns) — basically, codons not able to be translated into an amino-acid given the state of the gene processor[6], don't contribute to the generated resultant sequence in terms of sections it contains — with the exception of the special "Met" codon[7] that might or might not be retained in the resultant sequence even though it is automatically included as the first produced amino-acid in any legitimate gene sequence.*
2. *The entire sequence $[\Theta_n^*]^*$ is a kind of 3-Dimension molecule based on the chain of amino-acids it contains, and is technically referred to as a protein.*

□

*A **Ribosome** then, is any combination of transformers that can result in $[\Theta_n^*]^*$ when presented with just Θ_n as per the two intermediate transformer processes **Transformer 11** and **Transformer 12**, and whose overall processing algorithm is as depicted in **Figure 9.2** and its corresponding state machine as in **Figure 9.3**.*

*So, overall, a ribosome is any machine that can implement the combined transformer defined as in **Transformer 13**:*

[6] See **Figure 9.2** and **Figure 9.3**

[7] A START-codon also counted among genuine "exons" — coding codons

Transformer 13 (The **Protein Generator** (A Ribosome)). $\Theta_n \xrightarrow{O_{mRNA-encode}(\cdot)} \Theta_n^* \xrightarrow{O_{mRNA-translate}(\cdot)} [\Theta_n^*]^*$;
$\mathcal{U}([\Theta_n^*]^*) < \mathcal{U}(\Theta_n^*) = \mathcal{U}(\Theta_n) = n :$
$\psi_\Theta = \psi_{DNA} \quad \wedge \psi_{\Theta_n^*} = \psi_{mRNA} \quad \wedge \psi([\Theta_n^*]^*) = \psi_{amino-acids}$

And thus, we can finally merely call any machine capable of implementing the protein generator in ***Transformer 13*** *as a* ***Ribosome****.*

Chapter 10

Transformatics and Lu-Genome Expression System in Bio-Automata

> `Genome Expression: The collective expression profile of all genes within a genome, including coding and non-coding regions, across different conditions, cell types, or developmental stages. It is a systems-level view --- measuring how the entire genome behaves dynamically. This includes: mRNA expression of all genes, non-coding RNA activity, epigenetic regulation, chromatin accessibility and transcriptional networks. In essence, gene expression is local, while genome expression is global.`
>
> — **Microsoft Copilot**[24]

Away from using transformatics to make sense of the differences between organisms based on their genetic code sequences, an even more exciting application of the theory and practical methods of transformatics would be in conceptualizing, analyzing and explaining the important biological, physical, chemical, informational and mathematical concept of **genome expression**. We shall use a thought experiment and hypothetical cases leveraging exemplary genetic code modeled using special sequences that are treated as modal sequences at minimum.

First, we shall define a basic cipher that can encode the basic digits (of base-10), as some unique, but non-digit forms (essentially, with non-digit *glyphs*). This, so when we speak of a living organism — for example a pine-apple or a feline, we don't expect that in nature, or rather, that by physically dissecting the organism, that one shall find trapped or stored inside it, the literal genetic code symbols (such as those in ψ_{na}), but rather, that they shall find natural literal expressions or instances of genetic code material basis — such as the chemical base molecules that define nucleobases, or nucleotides, and also complex molecular structures such as transcribed amino-acids post-RNA translation of DNA expressions.

So, in our hypothetical genome expression model — the **Lu-Genome Expression System** (LGES), we are going to expect elements from our basic hypothetical DNA symbol set:

$$\psi_{\Omega} = \langle 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 \rangle \equiv \psi_{10} \tag{10.1}$$

Which, especially for curiosity's sake, but also, for creative and expressive reasons, is meant to allow us to explore a peculiar genetics model based on not just some four base symbols as is the case in ψ_{DNA}, but which allows us to look at any base-10 number expression as a potential genetic code sequence with **each digit expressing a distinct hypothetical nucleic acid**. For example, we might describe some fictitious organism — for simplicity's sake, perhaps just a virus that we shall name **the Euler Virus**[1] or just "veuler", as having its entire genome sequence encoded as just:

$$\Omega_{veuler} = 27182818 \tag{10.2}$$

However, that is the DNA-side of the story. As in real nature, we need a *different* alphabet or symbol set for expressing genetic code in its *intermediate form* — as genome expression typically proceeds via expression of DNA into mRNA first of all, and then finally into functions such as proteins. So, in our case, the intermediate expressions shall be expressed using the **Ozin Cipher**[2], which we are to describe hereafter.

10.1 The Ozin Genetic Code Cipher

With the background we have obtained thus far, note that the special **transcription level** expression of our genetic code originally expressed via ψ_{10}, shall be transcribed into the intermediate **ozin genetic code** that spans the symbol set ψ_{oz} that is mapped from the genetic code storage expression via a mapping as depicted in **Figure 10.1**.

[1] We called it that, simply because of the fact the associated genome sequence is based on the important natural physical constant — 2.718281828459, the **Euler number**, that is one of those irrational numbers that also happens to be the base of the natural logarithm and which arises naturally in many areas of mathematics, especially in calculus, complex analysis, and probability theory.

[2] More about this in the **Appendix**.

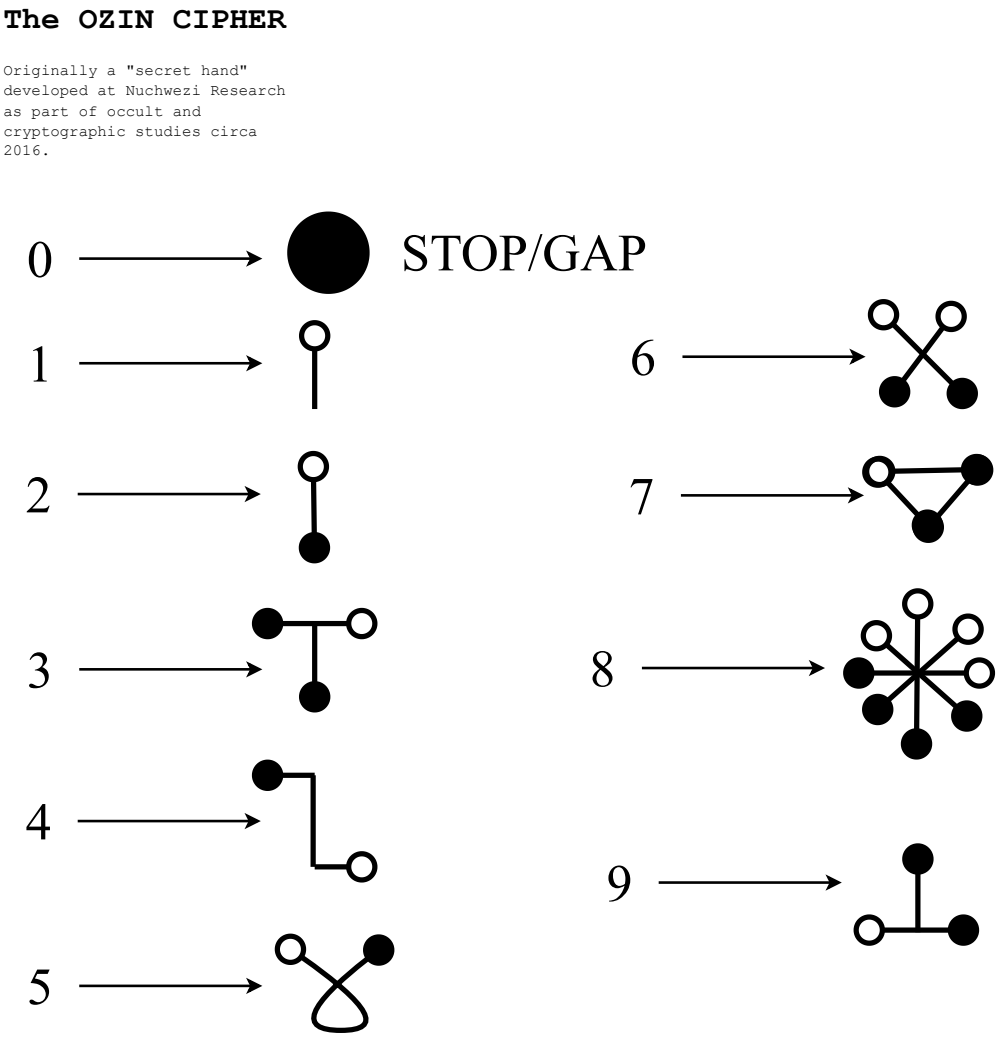


Figure 10.1: The OZIN Cipher[3], a mapping from Decimal Symbols (our base-Ω) to Symbols for Intermediate Genetic Code.

For simplicity's sake, we might interpret or transcribe from base-Ω DNA into our base-OZ RNA expression as such:

If for some organism such as that expressed via $\Omega_{veuler} = 27182818$, we wish to encode the equivalent intermediate [physical] expression, then we iterate through each symbol in the source code sequence, and re-write it as the equivalent symbolic structure in OZIN, and this, so as to center the ozin glyphs along a **backbone structure** of just a mere line that starts with a tiny dot and ends with the last ozin structure from the source sequence transcribed. Essentially, for our Euler Virus, it would render as something of the sort shown in **Figure 10.2**

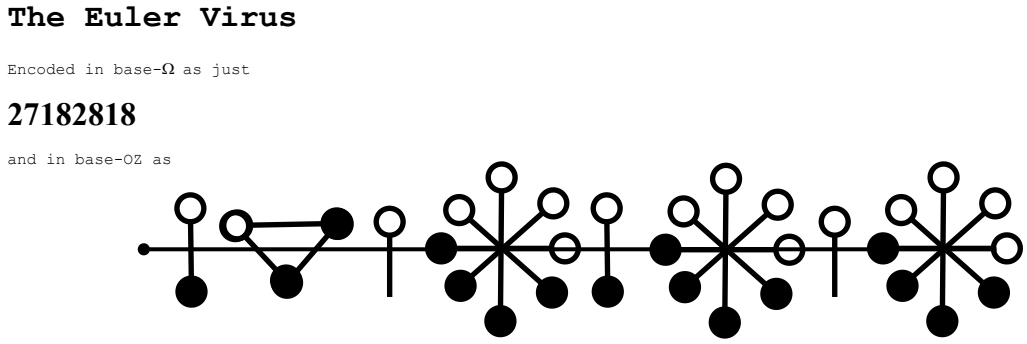


Figure 10.2: The equivalent OZIN expression of the hypothetical Euler Virus genome sequence.

Another organism, which we might just refer to as the **HiFinelle**, is actually based off of our favorite base-10 o-SSI — the **Hi-Fi o-SSI**[32], and thus, its corresponding genetic code at rest is as:

$$\Omega_{HiFinelle} = 8649137520 \tag{10.3}$$

And which, after we have it transcribed into intermediate OZIN genetic expression, shall appear as shown in **Figure 10.3**

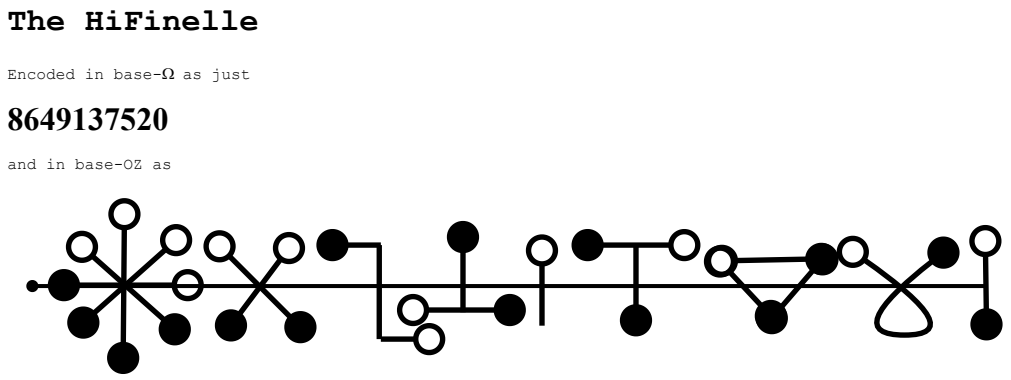


Figure 10.3: The equivalent OZIN expression of the HiFinelle genome sequence.

Of course, in our simplistic genome expression system, the occurrence of that last "0" symbol in the HiFinelle genome sequence tells us to STOP or just exclude that symbol with a GAP in case any other symbols follow thereafter. It is our STOP-codon equivalent.

Another genome sequence,

$$\Omega_3 = 0123026 \tag{10.4}$$

Might help illustrate that point — see **Figure 10.4**

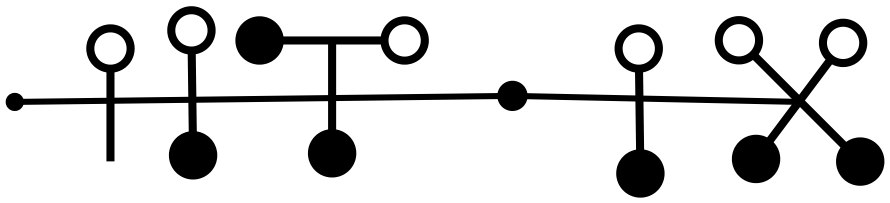


Figure 10.4: The equivalent OZIN expression of the Ω_3 genome sequence.

10.1.1 Ω-to-OZIN Genetic Code Transcription Rules

So, in general, we might sum up our gene transcription rules as such:

Algorithm 5 (The **Ω-to-OZIN Genetic Code Transcription Algorithm**). *Assuming we have an Ω-base encoded genetic code sequence Θ_Ω of some length $n > 0$*

1. *Initialize the transcription by constructing a backbone-structure that's essentially just a line with a dot at the head/start. This is the transcribed sequence $OZ(\Theta_\Omega)$.*
2. *For each symbol ω_i in Θ_Ω:*
   (a) *Use the transcription mapping: $\psi_\Omega \to \psi_{oz}$ as depicted in **Figure 10.1**, to encode ω_i in the equivalent form for the intermediate code.*
   (b) *If $\omega_i = 0$:*
      i. *If $OZ(\Theta_\Omega)$ was still empty:*
         A. *Simply skip to the next symbol in Θ_Ω (loop from **Step#2**).*
      ii. *Else:*
         A. *Append/Insert a GAP in $OZ(\Theta_\Omega)$*
         B. *Proceed to the next symbol in Θ_Ω (loop from **Step#2**).*
   (c) *Else:*
      i. *Place that encoded version of ω_i along the backbone structure, $OZ(\Theta_\Omega)$, at the next vacant position (i).*
      ii. *Proceed to the next symbol in Θ_Ω (loop from **Step#2**).*
3. *Truncate the backbone-structure after the final term in $OZ(\Theta_\Omega)$.*
4. *Return $OZ(\Theta_\Omega)$ as the final transcribed version of Θ_Ω .*

10.1.2 Examples of Ω-Genetic Code Transcription

So, now that we have the general transcription rules as in **Algorithm 5**, we can proceed to explore how to leverage them.

First, note that in **Chapter 5**, we have looked at not just the interesting idea of [genetic code] sequence complements, but have also explored several applications of the idea in **Section 5.2**, including the concept of the equivalence of sequences under the complement transform. Talking of which, assuming we had a special genome sequence that is made up of subsequences that are complements of each other (such as we might call **palindromic na-Sequences**) — e.g Θ_{pal} defined below, that is actually a palindrome of the two DNA codons — **ATG** (Methionine) and **CAA** (Glutamine) — the complements being **GTA** (Valine) and **AAC** (Asparagine)[35].

$$\Theta_{pal} = \langle A, T, G, C, A, A, A, A, C, G, T, A \rangle \tag{10.5}$$

If we express Θ_{pal} in our special numeric **base-Ω**, and then into the visual **base-OZ**, we shall get something like ${\Omega_{pal}}^3$:

$$\Omega_{pal} = \langle 1, 2, 3, 3, 2, 1 \rangle \tag{10.6}$$

And which, if we transcribe it into the intermediate OZIN code, we get:

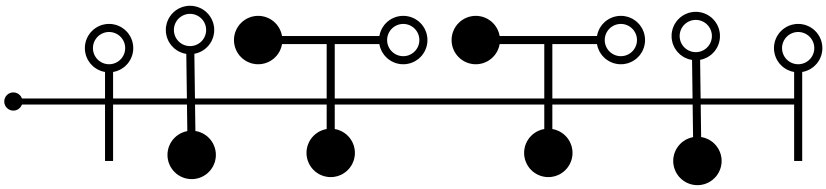


Figure 10.5: The palindromic sequence Ω_{pal} transcribed into base-OZ.

Now, notice something interesting here. In particular, if we compare the transcribed forms of Ω_{pal} and Ω_3, we shall notice that even though the two sequences had some differences — for example, Ω_3 contains the "0" symbol twice, including at the start, and yet, somewhat, because that STOP-codon is skipped during expression/transcription, so we note that the OZIN-expressions of the two sequences are somewhat similar — they both share a similar prefix structure, that upon inspection of their genetic code sequences, can be attributed to the 3-gram subsequence "123".

In fact, this later case is an interesting way to appreciate that even where two organisms, genes or proteins might exhibit significant differences in their appearance or even genetic code, and yet, upon close inspection, we might find some telling signs that they share some **common-traits**. In this case, we see

[3]Simplified, so we can easily visualize it in base-OZ

that the palindromic Ω_{pal} looks somewhat like the otherwise different Ω_3 — at least in their prefix structures after transcription (and possibly beyond!)

10.2 Genetic Code Sequences Processed as Modal Sequence Statistics

Now that we have finished looking at how to express genetic code from its base/storage-form (equivalent to DNA) into its intermediate-form (equivalent to RNA), we can then consider the matter of how that intermediate code eventually gets **translated** into the final organism[4].

So, in further advancement our thought experiment concerning genome expression (modeled using hypothetical bio-automata), we shall again utilize the OZIN cipher introduced in **Section 10.1**, but shall also introduce another formalism in the form of decoding position within a sequence (such as genetic code in this case), as information about how to express or process the associated symbols. This idea, of leveraging *subtle instructions* embedded in genetic code but which might not be explicit as the actual nucleic acid combinations (such as amino-acid encoding-codons), can be justified when we consider that the genetic code processor (such as the ribosome in our major context — see **Section 9.2**), has a way of interpreting sequences not just by what they contain, but also based on the context within which they exist. For example, though generally the appearance of the RNA codon, **AAA** would cause the ribosome to synthesize the corresponding amino-acid Lysine (see **Table 9.1**), and yet, if such a codon appears in a sequence where no earlier instruction compels the ribosome to START manufacturing a protein, the appearance of that codon shall be useless — or rather, it shall be interpreted differently merely because of being in the wrong context. We saw this in **Section 9**, and especially the Ribosome's protein-manufacturing process diagram (**Figure 9.2**) as well as the State Machine (**Figure 9.3**) help bring this out clearly. For example, if we modify the gene-program in **Equation 9.11** as such:

$$\Theta_4^* = \langle\langle A, A, A\rangle, \langle A, U, G\rangle, \langle U, U, A\rangle, \langle U, A, G\rangle\rangle \tag{10.7}$$

Then, where we would have expected the ribosome to manufacture protein whose structure includes the AAA/Lysine amino-acid, the result/effect of our modified sequence shall actually be different from what the original gene-program would have produced. In particular, since the START-codon, AUG (Methionine) appears after AAA, the effective protein to be synthesized would be as equivalent to just the program:

[4]We say "final organism" here, because, much as in reality, genome expression might be more complex than merely translating genetic code from DNA into RNA and then final proteins and tissue, and yet, conceptually, that is all there is to it!

$$\Theta_4^* = \langle\langle A, U, G\rangle, \langle U, U, A\rangle, \langle U, A, G\rangle\rangle \tag{10.8}$$

Which would produce the protein with just **UUA**(Leucine) in it! So, position of symbols or terms within genetic code matters, and does encode some important gene or even genome expression information that we can't take for granted, thus the following suggested system for how to express post-RNA genetic code sequences.

10.2.1 The Platonic Form Cipher and Position-Sensitive Genetic Code Translation

Just like we developed a system for how to map numbers (in base-10) to some non-trivial expressions that we equate to a kind of intermediate genetic code (the OZIN-encoded expressions), we shall similarly start by considering a way to map (still base-10) numbers to spatial-structures[5].

So, with the exception of "0", which again is treated as an instruction to 'skip' or 'stop' within our genome expression system[6], we shall adopt the mapping from numbers to spatial-structures as depicted in **Figure 10.6** in what we are calling the **PLATONIC Cipher** System[4].

[5]Especially because, and essentially that, when compared to the code that expresses them, proteins and the organism taken as a whole, *occupy more space* than does just the basic DNA code from which they are expressed.

[6]Perhaps, and as we shall adopt here, we shall interpret '0', especially *within* and not at the start or end of a sequence, to mean not just a GAP as was the case in **Algorithm5**, but as a JOINT. It somewhat makes biological/anatomical sense.

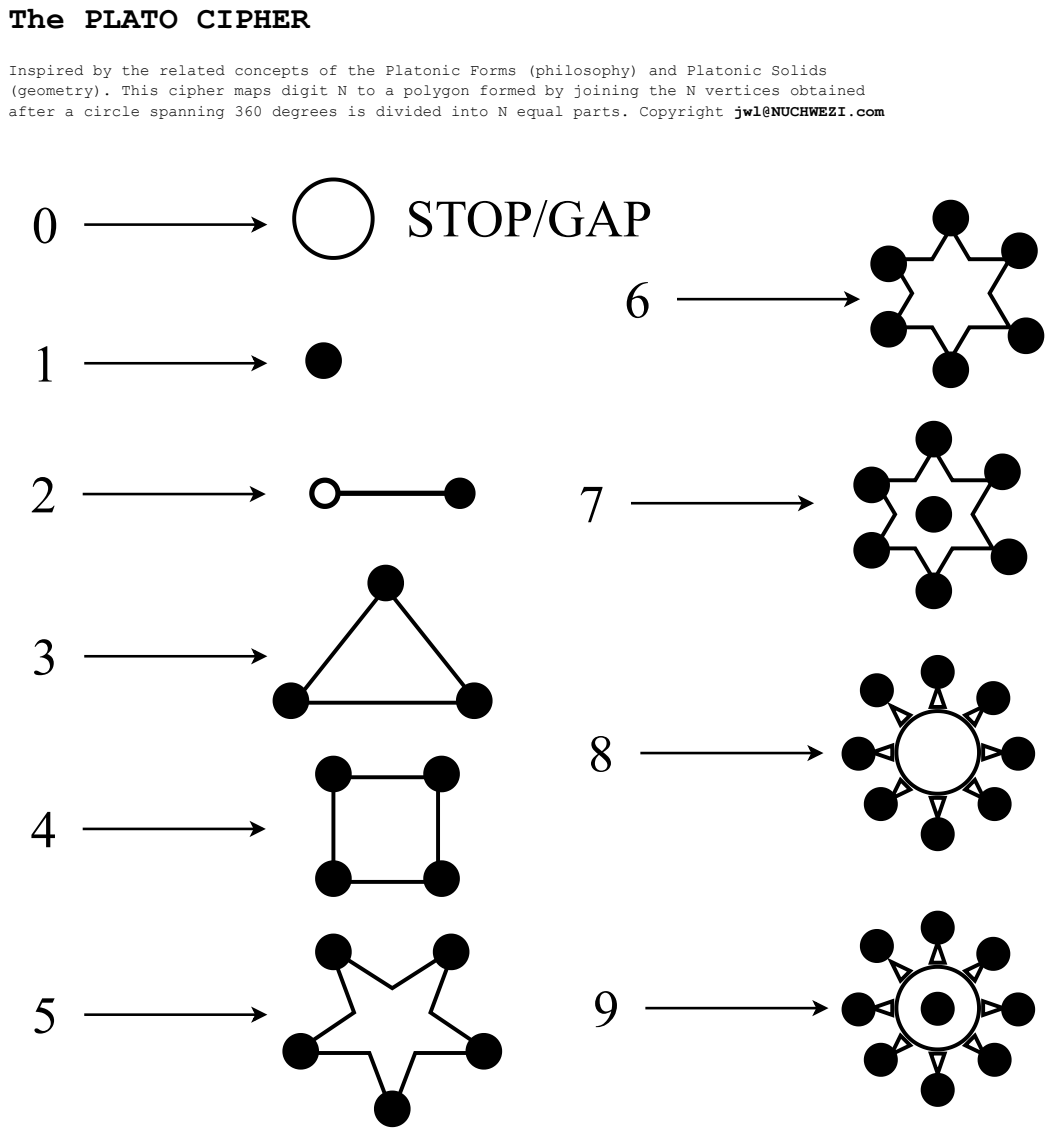


Figure 10.6: The PLATONIC Cipher[4] a mapping from Decimal Symbols (our base-Ω) to Spatial-Geometric Forms (ψ_{pf}).

How does it work?

10.2.2 ψ_Ω-to-ψ_{pf}: PLATONIC Form Genetic Code Translation Rules

So, in general, we might sum up our start to finish gene (and/or genome) expression with the post-transcription rules and process as such:

Algorithm 6 (The Ω**-to-PLATONIC Form Genetic Code Expression Algorithm**). *Assuming we have an* Ω*-base encoded genetic code sequence* Θ_Ω *of some length* $n > 0$.

1. *Initialize the expression with a transcription, by Using* ***Algorithm 5*** *to construct the* ***Intermediate Form Affix*** *(IFA) of* Θ_Ω *— essentially return* $OZ(\Theta_\Omega)$ *as the start of the* ***Final Genome Expression*** *(FGE)* $\boxed{OZ(\Theta_\Omega)}^*$ *— so that the final complete expressed(transcribed+translated) sequence shall be as:*

$$\boxed{OZ(\Theta_\Omega)}^* = OZ(\Theta_\Omega) \cdot \boxed{OZ(\Theta_\Omega)} \tag{10.9}$$

$\boxed{OZ(\Theta_\Omega)} : \mathbb{N} \times \psi_{pf}$ *is the suffix of expressed na-Sequence* Θ_Ω*, and is expressed in our genome expression system, as PLATONIC-Form encoded* ***modal sequence statistic*** *of* $OZ(\Theta_\Omega)$[7] *— the* ***PFA***[8]*, i.e.:*

$$\boxed{OZ(\Theta_\Omega)} = \boxed{\overset{>}{OZ}(\Theta_\Omega)}_{pf} \tag{10.10}$$

2. *So once we have the* ***IFA****,* $OZ(\Theta_\Omega)$*, proceed to use it to generate the rest of the sequence expression as follows:*

   (a) ***INITIALIZE*** *FGE,* $\boxed{OZ(\Theta_\Omega)}^*$ *as just:*

$$\boxed{OZ(\Theta_\Omega)}^* = \langle\langle OZ(\Theta_\Omega)\rangle\langle\rangle\rangle \tag{10.11}$$

This ensures, we have the prefix affix, ***IFA****, as the contents of the sequence start of* ***FGE*** *extended by a* ***still empty subsequence*** *— in expression form, this shall merely be a backbone structure with* ***IFA*** *as its head, and a terminal that is ready to have more units appended to it,* ***after a single GAP/JOINT extension to IFA****. So, it also means, at this juncture,* ***PFA*** *is just:*

[7] We render the **suffix of FGE** as $\overset{>}{OZ}(\Theta_\Omega)$ because it allows us to not express Θ_Ω naively at the genome or gene level, but also, so that, from an information-theoretic perspective, the completed genome expression is somewhat a statistical summary of what the genome sequence contains — the appearance of an animal or a plant or virus, depicting or expressing the summary of its DNA code. We do this then, by not merely doing a 1-to-1 translation of the intermediate Ozin-code expression $OZ(\Theta_\Omega)$, but instead computing its summary in form of $\overset{>}{OZ}(\Theta_\Omega)$, and then instead translating that into the final spatial/geometric forms as per ψ_{pf}. This is shall complete the entire processing/synthesis of sequence Θ_Ω, or rather our genome expression.

[8] Platonic Form Aspect

$$\boxed{OZ(\Theta_\Omega)} = \langle\rangle \tag{10.12}$$

(b) ***COMPUTE*** *the* ***IFA MSS****,* $\overset{>}{OZ}(\Theta_\Omega)$

(c) ***COMPUTE*** m*, the cardinality of the* ***IFA MSS****,* $m = \text{ш}(\overset{>}{OZ}(\Theta_\Omega))$

(d) ***COMPUTE*** *the* ***Platonic Form Encoding Key****(PFEK),* $\boxed{\psi_{pf}}_m$*, corresponding to* m *and based on the* ***SLSA Algorithm***[9]

(e) ***FOREACH*** *symbol/term* ω_i^* *in* ***IFA MSS****,* $\overset{>}{OZ}(\Theta_\Omega)$*:*

i. ***USE*** *the* ***symbol set translation mapping****:* $\psi_\Omega \rightarrow \boxed{\psi_{pf}}_m \rightarrow \psi_{pf}$ *where that final term corresponds to the mapping depicted in* ***Figure 10.6****, to translate* ω_i^* *into the equivalent* ***Platonic Form****,* $\boxed{\omega_i^*}$*, for the final expression as such:*

ii. ***IF*** $\omega_i^* \equiv 0$*:Operating on the current* ***FGE****:*

A. Append/Insert a JOINT onto ***FGE****,* $\boxed{OZ(\Theta_\Omega)}^*$

$$\boxed{OZ(\Theta_\Omega)}^* = \boxed{OZ(\Theta_\Omega)}^* \cdot \boxed{\omega_i^*}_{joint} \tag{10.13}$$

And equivalently/thus:

$$\boxed{OZ(\Theta_\Omega)} = \boxed{OZ(\Theta_\Omega)} \cdot \boxed{\omega_i^*}_{joint} \tag{10.14}$$

B. ***PROCEED*** *to the next symbol in* ***IFA MSS****,* $\overset{>}{OZ}(\Theta_\Omega)$ *(loop from* ***Step#2(e))****.*

iii. ***ELSE****:* ***RESOLVE*** *the correct* $\boxed{\omega_i^*}$ *for* ω_i^* *by using the relation*[10]

$$\boxed{\omega_i^*} = \rho_j \in \psi_{pl} + j \times \omega_i^* : j = \boxed{j} : I(\boxed{j}, \boxed{\psi_{pf}}_m) = I(\omega_i^*, \overset{>}{OZ}(\Theta_\Omega)) \tag{10.15}$$

[9]The **SLSA**, **Second Lu-Shuffle Algorithm** is well introduced and explained in **Section 8.2**, and it basically is about generating an exact anagram of the input sequence, corresponding to a particular partitioning index — in this case (and by **LGES** convention, denoted), m, the size of the **IFA MSS**

[10]Concerning how to actually arrive at the correct $\boxed{\omega_i^*}$ via the correct $\boxed{\psi_\Omega}_m$ for a given **IFA MSS**, refer to **Appendix B** that covers how to compute or look-up the necessary **platonic form encoding key**.

A. ***PLACE*** *that encoded version,* $\boxed{\omega_i^*}$*, along the existing* ***FGE*** *structure at the next vacant position (*$\text{ɹ}(\Theta_\Omega)+i$*). Expressible with another version of* ***Equation 10.13*** *as:*

$$\boxed{OZ(\Theta_\Omega)}^* = \boxed{OZ(\Theta_\Omega)}^* \cdot \boxed{\omega_i^*} \tag{10.16}$$

And equivalently:

$$\boxed{OZ(\Theta_\Omega)} = \boxed{OZ(\Theta_\Omega)} \cdot \boxed{\omega_i^*} \tag{10.17}$$

Where the special ***PFA*** *term,* $\boxed{\omega_i^*}$*, as per* ***Equation 10.15****, is basically the PLATONIC Form corresponding to position* j *in* $\boxed{\psi_{pf}}_m$ *— an* ***anagrammatized*** pf***-symbol set*** *relative to* m *and which is used to look-up the* ***correct indices*** *for the* ***corresponding*** *geometric structure components from* ψ_{pf} *the symbol set of the final form of the* ***PFA****, but also with the twist that the intermediate form* ω_i^* *is attached to that geometric structure* j *times — as depicted in example*[11] ***Figure 10.7****, with each expression of* ω_i^* *attached to one of the* j ***appendage spots*** *on the shape for* $\boxed{\omega_i^*}$*.*

B. ***PROCEED*** *to the next symbol in* ***IFA MSS****,* $\overset{>}{OZ}(\Theta_\Omega)$ *(loop from* ***Step#2(e))****.*

3. ***TRUNCATE*** *the final-structure after the final term in* ***FGE****.*

4. ***RETURN FGE****,* $\boxed{OZ(\Theta_\Omega)}^*$ *as the final transcribed and translated version of* Θ_Ω*. The culmination of our conceptual genome expression.*

□

[11] It shall be interesting to note, that based on the **PFEK** by **SLSA** — see **Table B.2**, that assuming we had [an actually *strange*] organism whose equivalent Ω-encoded DNA sequence were just the sequence Θ = **666**, its corresponding **IFA** characteristic would be just $\hbar(\overset{>}{\Theta}) =$***63***, the corresponding **MSS** just $\langle \mathbf{6} \rangle$, and so its **PFEK** look-up key, $m = \text{ɹ}(\overset{>}{\Theta}) = 1$, so that, its corresponding $\boxed{\psi_{pf}}_1 = \langle 4 \rangle$, and so that for its **BODY** component in the associated full genome expression, the entire **PFA** would just be $\boxed{\omega_1^*} \implies 4_{pf} \otimes 6_{oz}$ — which is exactly what we see in this picture in **Figure 10.7**!

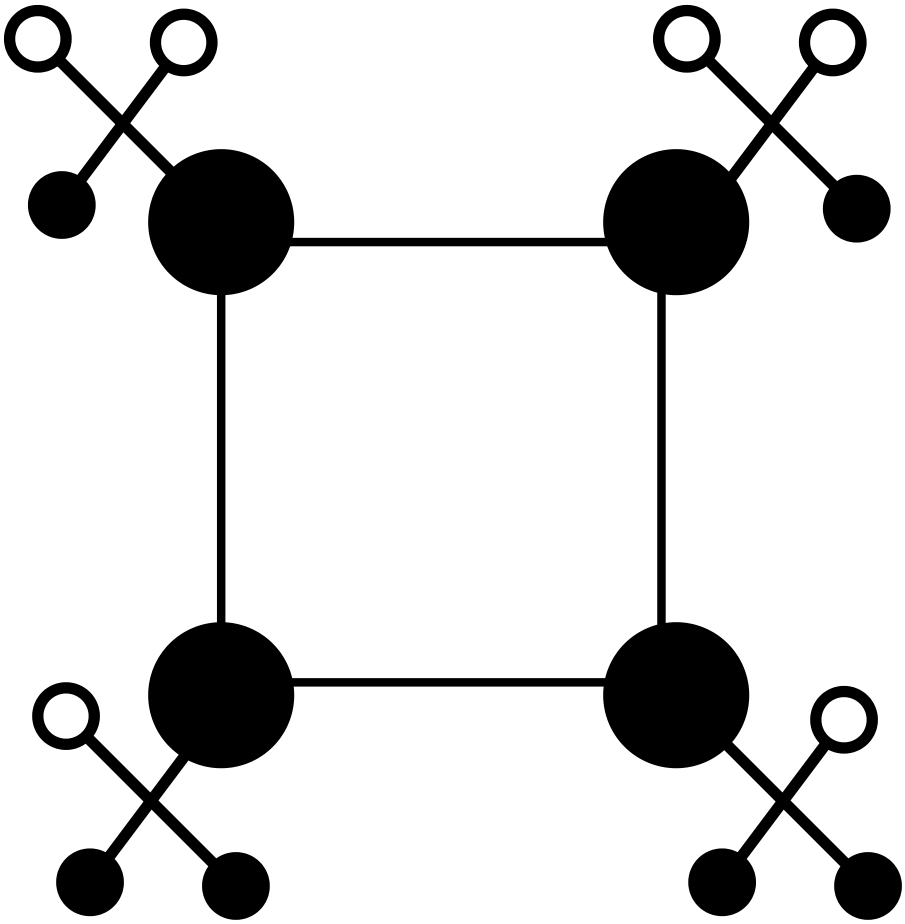


Figure 10.7: An example of a single **amino-acid** or rather, the **PFA** component of a **FGE** equivalent to one **amino-acid** segment expressed using the **LGES** system. This particular expression corresponding to a rendering of the configuration $4_{pf} \otimes 6_{oz} = 4_{pf} + 4 \times 6_{oz}$.

After looking at some examples of actual genome expressions rendered using the **LGES**, perhaps most reviewers of **Algorithm 6** shall come to appreciate that it is not only non-trivial, but is likewise plausible given the sensibilities of how actual expression occurs in nature/reality[12] — see a natural expression as in **Figure 1.1**, and that it encapsulates the ideals of both distinctiveness but also diversity for the minutest of organic creatures that are just as basic as mere genetic code with a basic encapsulating body (such as viruses), but also complex organisms belonging to distinct families of prokaryotes and eukaryotes.

10.2.3 Examples of Complete Lu-Genome Expressions

TODO: FOR THE REST OF THIS CHAPTER... review all LGES and x-LGES diagrams as per highlighted tricky step in algorithm to ensure they are consistent

[12]That said, it shall be up to others — botanists, zoologists, molecular biologists, virologists, etc. to either confirm or refute whether or not **LGES** does indeed echo the reality in nature or whether it is a theory only applicable to a subsection of reality — *which if not all?*

with rules. BUT also review the translation/transcription equations so that they reflect the correct **IFA MSS** to **PFA** logic as per the new algorithm for the **PFA** in particular — this has already been fixed for those in the ORIGINAL/Appendix, just review the ones in this chapter then.

We get to see the first example of the output of processing some na-Sequence expressed originally in base-Ω such as for Ω_3 from **Equation 10.4**, in its final expression form as in **Figure 10.8**[48] — more importantly, since we have already seen its **IFA** (the **FGE** prefix) in **Figure 10.4**, we note that we essentially see the update as the result of expressing the entire **FGE** via the **IFA MSS** equivalent to:

$$\overset{>}{\Omega}_3 = \langle 0, 2, 1, 3, 6 \rangle \implies \boxed{OZ(\Omega_3)} \approx \langle GAP, 7, 8, 6, 3 \rangle \tag{10.18}$$

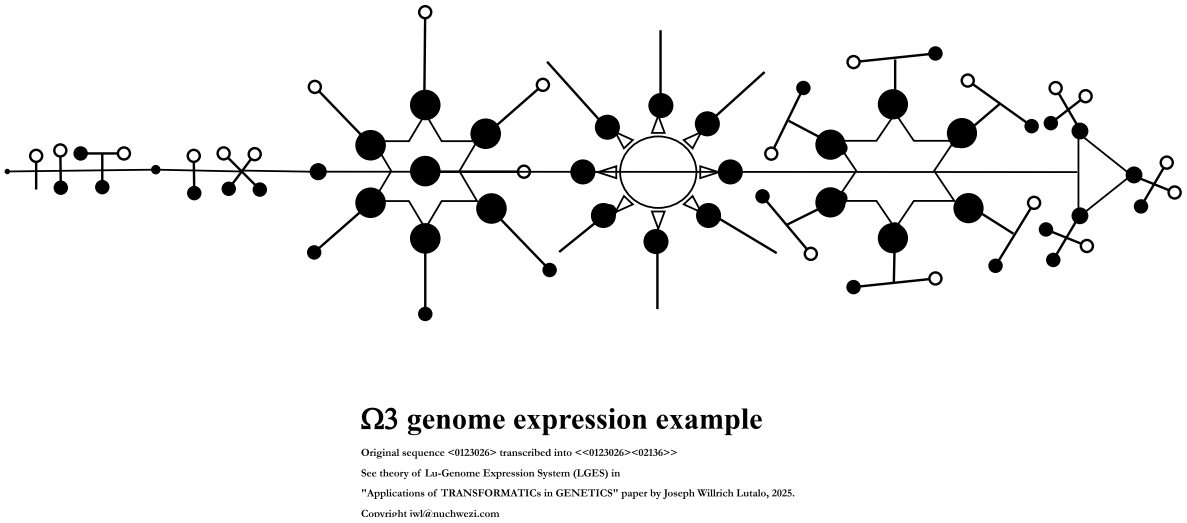


Figure 10.8: The equivalent LGES complete genome expression of the Ω_3 genome sequence (from **Equation 10.4**)

Though we won't delve into exploring all kinds of possible *interesting* scenarios with regards to the **LGES**, we can look at one other example and then move on. From the example hypothetical genome sequences we have encountered, perhaps let us revisit **Equation 10.2** and see what the **Euler Virus** might look like when fully expressed in our system.

So, first of all, note that we already did the first-leg of **Algorithm 6**, and we have seen what the **FGE** prefix — the **IFA**, would look like in **Figure 10.2**[13].

[13]Talking of which, for the well-read reader, and especially students and explorers of not just esoteric languages but especially occult languages and systems — perhaps better we talk of **Occult Philosophy**[49], not only the coincidence that our OZIN-expressions look indeed *occult* — such as one might find utilized in occult systems such as John Dee's Enochian[47], but also that both the **IFA** and **PFA** — the later, expressed in a somewhat more familiar (because the polygons at the core of the PF-cipher are indeed commonplace in both mathematics and mysticism) language, but ultimately, the experienced observer might appreciate that complete **LGES** expressions/figures such as **Figure 10.8** look much like the traditional/ancient mystical glyphs known as **Mandalas** to modern scholars such as Carl G. Jung that enjoyed these subjects[50] and perhaps, and as the author himself has explored in works such as the novel "Rock 'N' Draw"[51], these figures so much remind one of **Voodoo Veves**! Perhaps, in a later work on either philosophy or mysticism by the author — see earlier works of this kind in [52], we shall want to take time off and properly explore this exciting dimension of human expression and culture, especially since, unlike what might have ever come before, our symbolic programs, unlike voodoo veves or chaos magick sigils, are rooted in two objective fields of inquiry — mathematics and biology. And no, though the acronym **IFA** reminds one of the *Ifa* esotericism of Western Africa, and yet, the author didn't have that in mind while working on the LGES at first.

So, like we did for Ω_3 in the last exploration, let us start by looking at what the numeral-equivalence of the processed Ω_{veuler} would be like. For brevity, we can sum up the entire production/transformation that leads to the complete **FGE** with correct prefix and suffix, and thus $\boxed{OZ(\Omega_{veuler})}^*$ as:

Transformation 18. $\Omega_{veuler} = \langle 2,7,1,8,2,8,1,8\rangle \rightarrow \hbar(\overset{>}{\Omega}_{veuler}) = \boldsymbol{83221271} \rightarrow \overset{>}{\Omega}_{veuler} = \langle 8,2,1,7\rangle \implies \boxed{OZ(\Omega_{veuler})} \approx \neg\langle 8,2,1,7\rangle_{pf} \implies \boxed{OZ(\Omega_{veuler})}^* = \langle 2,7,1,8,2,8,1,8\rangle_{oz} \cdot \boxed{\langle 1,7,8,2\rangle}_{pf}$

Note that in that derivation depicted in **Transformation 18**, one of the most tricky parts is computing that FGE suffix, $\boxed{OZ(\Omega_{veuler})}$, the **PFA** — which is formalized/defined by *especially*[14] **Equation 10.15**. So, once that is done, and we have our final numeral-**FGE** as such:

$$\boxed{OZ(\Omega_{veuler})}^* = \langle 2,7,1,8,2,8,1,8\rangle_{oz} \cdot \boxed{\langle 1,7,8,2\rangle}_{pf} \tag{10.19}$$

Of course, concerning **Equation 10.19**, note that, or rather, keep in mind that concerning the **PFA** component, $\boxed{OZ(\Omega_{veuler})}$, the corresponding associated sequence in numeral form is the **IFA MSS**, $OZ(\overset{>}{\Omega}_{veuler}) = \overset{>}{\Omega}_{veuler} = \langle 8,2,1,7\rangle$. And thus, using necessary transcription and translation ciphers as well as the multiplication rules required by **Equation 10.15** when treating of the **PFA**, we then generate the final genome expression for the **Euler Virus** under the **LGES** that looks as in **Figure 10.9**.

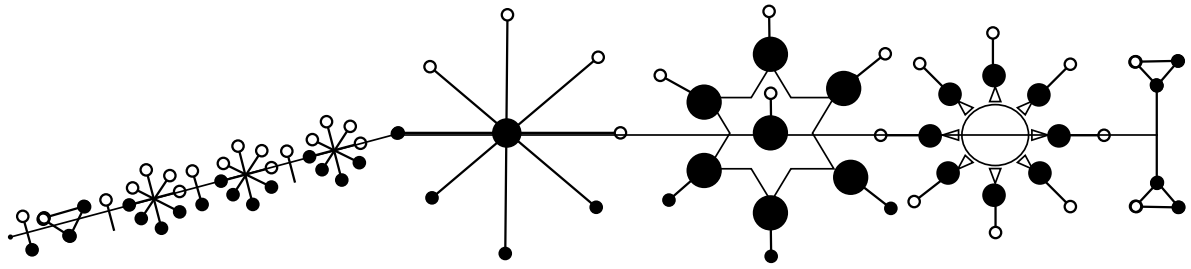


Figure 10.9: The equivalent LGES complete genome expression of the Ω_{veuler} (Euler Virus) genome sequence (from **Equation 10.19**)

Now, concerning that rendering of the genome sequence expressed in **Equation 10.19**, it might be worth noting that unlike the earlier examples of **LGES** expressions, we have taken the liberty to exploit that special joint that kick-starts the **PFA**, or rather, which separates the **IFA** from the **PFA** in a **FGE** expression. So, just like an organism with a physical joint might sometimes appear

[14] **Equation 10.14** is what covers the special symbol '0', that is treated as a JOINT/GAP in the **PFA** and as STOP/GAP in **IFA**.

as angled or curved, so we have chosen to depict the Ω_{veuler} with its one joint[15] well utilized. However, and important to note; it is not just joints that might appear different across different renderings of the same basis genome sequence. So, in **Figure 10.10**, we see another variation of the same sequence, with readily noticeable differences being the IFA-PFA joint in its neutral position, and then the second node in the PFA (corresponding to $7_{pf} \otimes 2_{oz}$ or rather $7_{pf} + 7 \times 2_{oz}$) with the 2_{oz} appendages depicted mostly aligned horizontally.

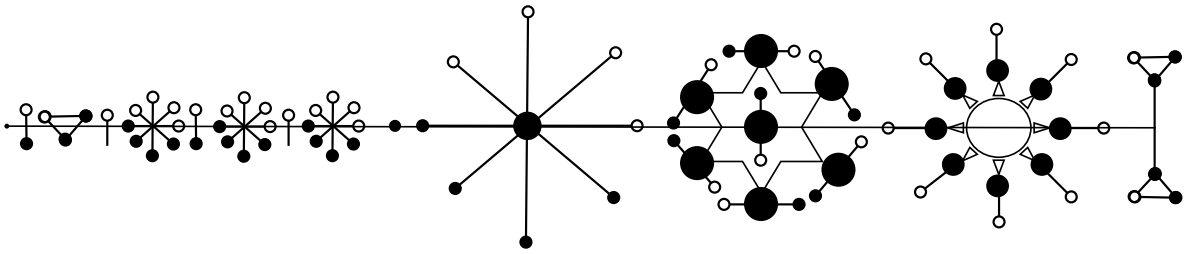


Figure 10.10: The **alternative rendering** of the same sequence as Ω_{veuler}.

Concerning **Figure 10.10**, the alternative rendering of the same sequence as Ω_{veuler} first depicted in **Figure 10.9**, we see that underlying the variations is how the *degrees of freedom* in rendering the genome have somewhat been exploited. Notice how the joint and appendages are placed or oriented about the platonic forms that mostly remain unchanged across the two renderings — except perhaps in scale. And talking of scaling nodes, in the future, we might want to explore whether in such a system as **LGES** in which the expression is sensitive to measures such as the **IFA MSS**, whether it might make sense to scale each subsequent pf-node in the **PFA** based on its position within the sequence? Though we didn't have such a rule included in **Algorithm 6**, and yet, our example renderings here have somewhat taken advantage of that plausible logic.

That said, note that we started this work with a photograph of a pineapple ready to harvest (see **Figure 1.1**), and though the associated discussion[16] by the author did bring up the matter of how it might inspire a genome expression system in which the prefix is the actual DNA that renders the rest of the organism (fruit, plant-base and roots in this case of the pineapple), it is left as an exercise to the reader to decide whether example genome sequence expressions/renderings such as in **Figure 10.9** depict this pineapple-inspired philosophy.

[15]Definitely, and interesting to explore, a genome sequence with multiple JOINT points in it might be how an organism with say appendages such as legs, hands and a head that can move independently of the core/thorax of the body is possibly arrived at. The interested student should go ahead and explore more non-trivial Ω-base sequences just to see what's possible.

[16]Checkout video/mini-lecture on the matter at author's YouTube: `https://youtube.com/shorts/yTJzBEpw7Z4`

NOTE:
On **Why Place-Notation Expression Systems Matter**

As a further justification for why we chose to use a place-notation sensitive genetic code expression system in our conceptual genome expression thought experiment, consider that, just like with modern numbers — of which the Arabic number system is considered one of the best since ancient times, the appearance of a symbol within an expression is interpreted not just by what it quantifies, but also by where in the expression it is placed.
For good measure, a few enlightening quotes from the priceless work on the evolution of number systems since ancient times, by Raymond S. Nickerson, shall help cement this:

> Aristotle attached considerable significance to the fact that human beings can count: it is this ability, he claimed, that demonstrates our rationality.
>
> As Brainerd (1973) has pointed out, in most Western nations today we expect students, by the time they reach their early teens, to be much more competent with numbers than was an educated adult Greek or Roman of 2,000 years ago. This fact arises in no small measure from the elegance and power of the scheme that we now use to represent numbers.
>
> In short, a price of the increased abstractness of the Arabic system is an obscuring of some of the key principles on which numbers are based. To an ancient Egyptian, the fact that 3+4=7 was apparent from the relationship between the Egyptian number representing 7 and those representing 3 and 4. There is no hint of such a relationship, however, when this equation is expressed in Arabic notation. The price has been worth the paying, however.
>
> If you invented a unique symbol to represent each of the possible quantities of stones (including the case of no stones) up to that of your standard and decided to use the position of that symbol to indicate the pile to which it refers, you would have what we refer to today as a place-notation system for representing numbers.
>
> The relative ease with which one can count or compute depends to no small degree on the characteristics of the system that is used to represent number concepts.
>
> —**Raymond S. Nickerson**, *Counting, Computing, and the Representation of Numbers*, 1988[53]

And finally, it shall be worth noting that this notion also underlies the present author's use of the **Position-Index measure** ($I(\omega_i, \Theta)$), first formalized in the Base-36 paper[54], and which, among many important applications it has, is its use in generalizing the computation of decimal values of arbitrary number expressions in any base, its use in transformatics to express sequence summaries using modal sequences[9], etc.

10.3 The Extended Lu-Genome Expression System (x-LGES)

If you look at the genome sequence expressions that we have encountered in **Section 10.2.3**, and compare that to the case of our pineapple in **Figure 1.1** that opens the introduction section, you shall notice that, just in case we assume our **LGES** protocols to be sufficient for the expression of not just bits of an organism, but also relevant to the expression of a complete organism, then, it might seem like something is still missing, and thus the **Extended Lu-Genome Expression System** (x-LGES).

So, what exactly is missing from our genome expressions thus far? We shall call out a few telling points to help drive the point home:

1. As has been stressed already, and as is the case that we wish our science to not be constructed in ivory towers alienated from facts and reality in nature, we must borrow a leaf from how actual DNA is utilized to express organisms in reality. For example, it is not that one genome expresses the hands, another the heart, and yet another the eyes! Well, of course, in a particular organism's DNA, there shall be specific subsequences that best define the expression of particular features or aspects of an organism, however, overall, a single, complete genome sequence is and should be sufficient for the complete expression of an entire organism with all its various attributes and distinct aspects well catered for.

2. When we consider the way the original **LGES** algorithm is specified — **Algorithm 6** being the one that combines both transcription and translation into a single process, we shall note that, the final output of that process — as depicted in **Equation 10.9** for the general case, and as in **Equation 10.19** for one particular organism's expression (in *numeral form*), we notice that, especially when approached from the point of view of the fully-processed rendering of the **FGE**, $\boxed{OZ(\Theta_\Omega)}^*$, some potentially relevant information could be considered to be missing from that expression at that point. In particular, when we have the **PFA** expressed in its platonic form, it might not readily be possible to discern the vital $\overset{>}{OZ}(\Theta_\Omega)$ aspect since the actual **PFA** is a combination (see **Equation 10.15**) of that **IFA MSS** in its OZIN form as well as its translation into complementary Platonic Form expressions via the **PLATO CIPHER**.

3. Finally, and again, returning to nature as our authority on genome expression, we note that most organisms — plants or animals, typically manifest in a **PREFIX-THORAX-SUFFIX/HEAD-BODY-ROOT** form. Looking at our LGES renderings in **Section 10.2.3**, we see that the **IFA** aspect properly caters for the "PREFIX/HEAD" component — it somewhat con-

tains the entire [genome] sequence just encoded in base-OZ[17]. Then we have the "THORAX/BODY" bit. It feels plausible to equate that with the protracted spatial-expressions of the **PFA**. But what of the "SUFFIX/ROOT" component? And talking of suffixes in natural genome expressions; we can appreciate that for humans, birds and most animals we can talk of **legs**; for aquatic animals especially, such as fish, we might talk of **tail-fins**; for insects and all various kinds of arthropods, we can not only talk of a body form that's generally made up of chained bits or adjoined segments such as in our **LGES**, but that they generally terminate in a kind of suffix we might consider to be **an abdomen**. And then finally for plants, it's obvious they typically terminate in **roots** — and especially for the case of food-crops, we find that these roots contain sufficient *bio-information* to allow us to reproduce the original plant organism from just those suffixes[18].

So, with those observations and criticisms of the original **LGES**, we can provide a solution in form of the following algorithm that extends **Algorithm 6**:

10.3.1 ψ_Ω-to-(ψ_{oz}-ψ_{pf}-ψ_{oz}): The x-LGES Algorithm

To render a complete genome expression using this extended algorithm, and starting from the original genome sequence encoded as a base-Ω expression, we proceed as such:

Algorithm 7 (The **x-LGES Algorithm**). *Assuming we have an Ω-base encoded genetic code sequence Θ_Ω of some length $n > 0$.*

1. *Initialize the final expression by constructing the initial part of the **FGE** using **Algorithm 6** to construct the **Intermediate Form Affix** (IFA) of Θ_Ω **and** the **Platonic Form Aspect (PFA)** — so that at this point, the complete expressed(transcribed+translated) sequence shall be as:*

$$\boxed{OZ(\Theta_\Omega)}^* = OZ(\Theta_\Omega) \cdot \boxed{OZ(\Theta_\Omega)} \tag{10.20}$$

2. *Extend that initial **FGE** with just **a single GAP/JOINT extension**, so that we then have the **FGE** as such:*

$$\boxed{OZ(\Theta_\Omega)}^* = \boxed{OZ(\Theta_\Omega)}^* \cdot \boxed{\omega_i^*}_{joint} \tag{10.21}$$

And equivalently:

$$\boxed{OZ(\Theta_\Omega)}^* = OZ(\Theta_\Omega) \cdot \boxed{OZ(\Theta_\Omega)} \cdot \boxed{\omega_i^*}_{joint} \tag{10.22}$$

[17] Interesting to say, "the brains" of the expressed organism, especially since it has the complete genome almost unaltered, and in its *actionable form*.

[18] Though not all root-crops might readily be grown from their root tubers, some good examples from observation and experience author has obtained practicing domestic crop-husbandry include irish potatoes, yams and ground-nuts!

3. *Then, using the **IFA MSS** from **Step#1/Algorithm 6**, that is* $\overset{>}{OZ}(\Theta_\Omega)$, *as the **Genome Expression Suffix (GES)** extend the current **FGE** so that*

$$\boxed{OZ(\Theta_\Omega)}^* = OZ(\Theta_\Omega) \cdot \boxed{OZ(\Theta_\Omega)} \cdot \overset{>}{OZ}(\Theta_\Omega) \tag{10.23}$$

Which, if qualified with the necessary encoding bases is as such:

$$\boxed{OZ(\Theta_\Omega)}^* = OZ(\Theta_\Omega)_{oz} \cdot \boxed{OZ(\Theta_\Omega)}_{pf} \cdot \overset{>}{OZ}(\Theta_\Omega)_{oz} \tag{10.24}$$

*And if we express this as the update to **Equation 10.25**, we have:*

$$\boxed{OZ(\Theta_\Omega)}^* = \boxed{OZ(\Theta_\Omega)}^* \cdot \boxed{\omega_i^*}_{joint} \cdot \overset{>}{OZ}(\Theta_\Omega)_{oz} \tag{10.25}$$

*Which tells us that the final **FGE** is the original **FGE** extended with just one extra JOINT and the **GES/IFA MSS** encoded in base-OZIN.*

4. *Truncate the final-structure after the final term in **FGE**.*
5. *Return **FGE**,* $\boxed{OZ(\Theta_\Omega)}^*$ *as the final transcribed and translated version of* Θ_Ω. *The culmination of our extended genome expression system.*

We shall come to appreciate this updated protocol in **Algorithm 7**, by taking a few of the **FGE** examples we looked at earlier, and re-render them using the **x-LGES** to see what difference it makes.

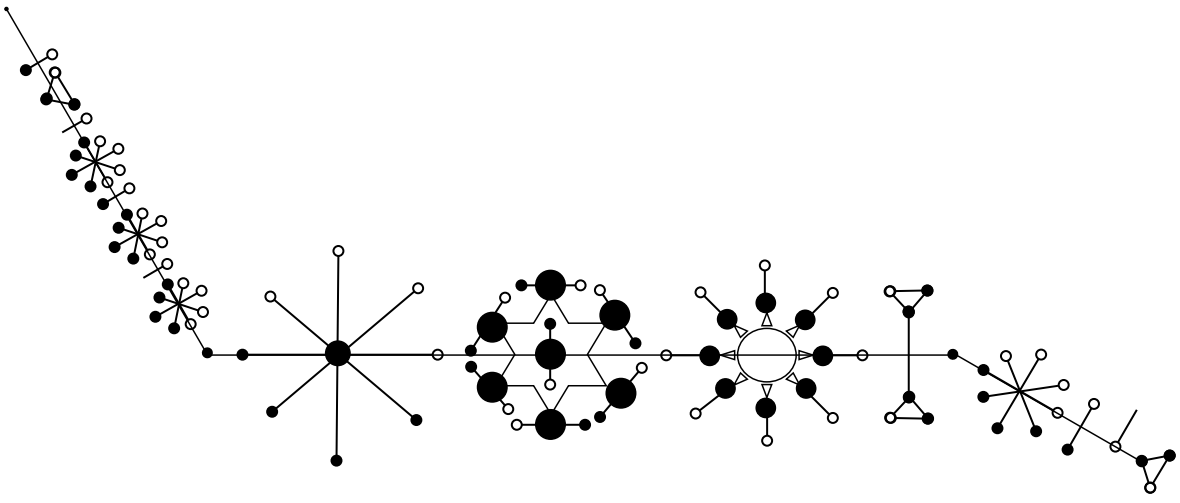


Figure 10.11: The **x-LGES form rendering** of the Ω_{veuler}, the **Euler Virus**, now with HEAD, BODY and ROOT segments.

The first example is with the notorious **Euler Virus** originally defined in **Equation 10.2**, and this rendering shown in **Figure 10.11** shall start to paint the new picture for you, if you compare it to the original OZIN-expression of its equivalent **mRNA** as shown in **Figure 10.2**, and then that same organism as

the basic **LGES**-form in **Figure 10.10** which shows the full genome expression minus the extension **GES** SUFFIX/ROOT.

10.4 Actual DNA Sequences Expressed in the Extended Lu-Genome Expression System

So, now that we have finished looking at what could happen with genome expression under the hypothetical LGES protocols, one might wonder, and correctly so, how might we use that system to attempt to visualize actual na-Sequences such as DNA or RNA sequences or subsequences?

Concerning working with the LGES or x-LGES as it is at the moment, the entry point barrier is that the system is defined for sequences expressed in base-10 only. That means, unless your sequence is expressed using symbols from ψ_{10}, there is no direct way to leverage the theory we have just presented. So, that's where we need put our attention first, with regards to processing DNA or RNA in this thought experiment.

Good enough, we have had extensive background concerning transformation of sequences across symbol sets in earlier chapters, and so, we can just re-iterate some points here.

- We know that any na-Sequence shall span ψ_{na}, which has cardinality 5 (refer to **Law 1**), and yet, $\mathit{ɯ}(\psi_{10}) = 10 > \mathit{ɯ}(\psi_{na})$.
- For either DNA (ψ_{DNA}) or RNA (ψ_{RNA}), their corresponding symbol set cardinalities are just **4** and **4** respectively, which still doesn't map well to the base-10 or rather ψ_{10} symbol set that the Lu-Genome Expression System requires.

So, using sequences we have already encountered, such as Θ_4^* in **Equation 10.7** — an RNA sequence, or its original DNA form (as in **Equation 9.9**) how, should we go about expressing it in equivalent or sufficient Ω-form?

10.4.1 Converting Between Nucleic Acid Code and Numeric Codes

First, given the constraints we have seen above, one plausible solution might be to start with a naive[19] encoding that merely maps every element in ψ_{DNA} to a corresponding term in ψ_{10} **based not on the quantity of the symbol — since na-Sequences aren't made of 'literal numbers', but rather, based on the position of the symbol within the DNA symbol set**. So that, we have the following basic encoding-transformer:

[19]We think it is naive, because, using the position-index operator $I(\omega, \Theta)$ in the encoding definition, it isn't clear how we shall ever resolve for the digit '0' given DNA has only 4 distinct symbols.

Transformer 14 (ψ_{DNA} to ψ_{10} **DNA-Decimal Encoder**).
$\Theta \xrightarrow{O_{dna-dec-encode}(\cdot)} \Theta^*; \quad \forall \omega \in \Theta : I(\omega, \Theta) = k, \exists \rho \in \Theta^* : I(\rho, \Theta^*) = k \implies \rho = I(\omega, \psi_{DNA})$

And so, using that **Transformer 14**, we can process any DNA sequence and map it to corresponding Ω-form (or rather *decimal form*) as such:

$$\begin{bmatrix} A & C & G & T \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

With that basic mapping, and our example DNA sequence from **Equation 9.9**) we see that the corresponding decimal-encoded DNA sequence shall merely be as in the following transformation:

Transformation 19. $\Theta = \langle A, T, G, A, A, A, T, T, A, T, A, G \rangle \xrightarrow{O_{dna-dec-encode}(\cdot)} \langle 1, 4, 3, 1, 1, 1, 4, 4, 1, 4, 1, 3 \rangle$

Essentially, the symbol set for DNA sequences encoded in decimal shall only span the special symbol set we might refer to as **DNA-Digits Symbol Set**, and which we might denote as ψ_{DD}, and which is just:

$$\psi_{DD} = \langle 1, 2, 3, 4 \rangle \tag{10.26}$$

So, with ψ_{DD} and knowing the mapping from DNA to Decimal, we could also work backwards from decimal sequences back into DNA sequences. So, for example, assuming we took the interesting **Hi-Fi o-SSI** from **Equation 10.3**, and wanted to map it to an equivalent DNA sequence, we might proceed by combining the consequences of **Theorem 4** and **Transformer 14**. However, as warned earlier on, **Transformer 14** is somewhat *weak* when dealing with decimal-to-DNA decoding, especially because not all symbols in ψ_{10} shall have meaningful equivalent terms in ψ_{DNA}. So, better we start by solving that problem instead.

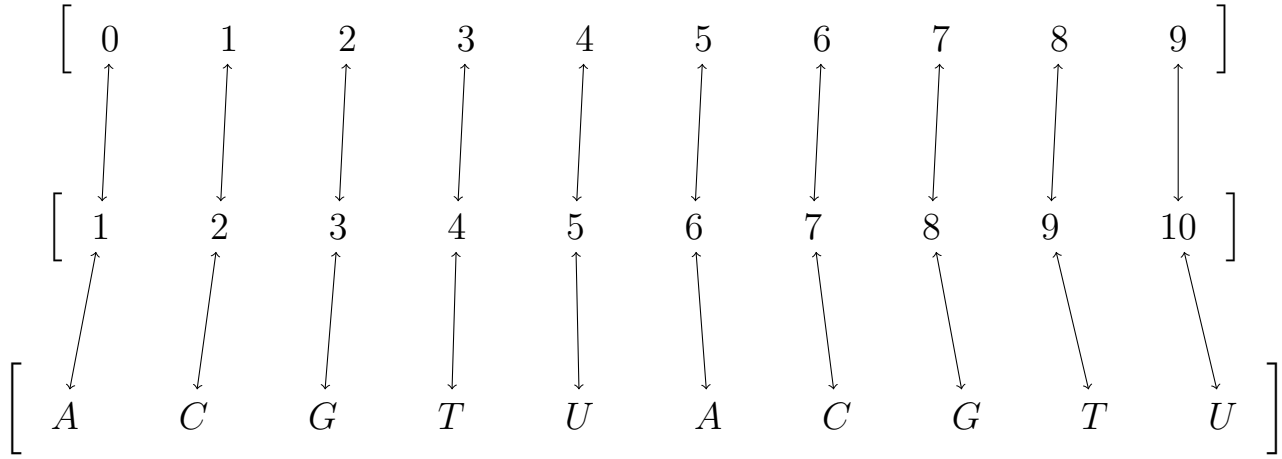
Thus, with some *necessary tweak* — that, instead of mapping from ψ_{DNA} to ψ_{10} and vice-versa, we take ψ_{na} which has exactly 5 elements and which is a superset of ψ_{DNA}[20], and then map any symbol from ψ_{na} to ψ_{10} and vice-versa as per the following non-trivial transformer:

Transformer 15 (ψ_{na} to ψ_{10} **na-to-Decimal Decoder**).
$\Theta \xrightarrow{O_{na-dec-decode}(\cdot)} \Theta^*;$
$\forall \omega_i \in \psi_{na} : I(\omega_i, \psi_{na}) = i \implies \exists d_j \in \psi_{10} : I(d_j, \psi_{10}) = j = i \quad \vee \quad I(d_j, \psi_{10}) =$

[20]Refer to **Definition 3**

$\mathbb{u}(\psi_{na}) + i \implies \omega_i \leftrightarrow d_j$
$\wedge \quad \forall \alpha_k \in \Theta \quad \exists d_k \in \Theta^* : I(\alpha_k, \Theta) = I(d_k, \Theta^*) = k \quad \wedge \quad I(d_k, \psi_{10}) = I(\alpha_k, \psi_{na}) \vee$
$I(d_k, \psi_{10}) = \mathbb{u}(\psi_{na}) + I(\alpha_k, \psi_{na}) \implies \alpha_k \leftrightarrow d_k \quad \square$

So that, by **Transformer 15**, we arrive at complete decimal-to-DNA and decimal-to-RNA or generally decimal-to-na-Sequence symbol encoding-decoding is as shown in the following diagram:



Of course, the necessary tweak here, especially for sequences target sequences specific to either DNA or RNA and not both, the trick terms 'T' and 'U' need to be dealt with as necessary — possibly with a basic swapping of all 'U' with 'T' for an na-Sequence targeting a DNA processor, and vice-versa for those targeting strict RNA processors.

Also, that mapping offers us a basic protraction na-Sequence that is a basic multiplication of the psi_{na} symbol set by 1. Because it maps nicely to terms in ψ_{10}, both sequences having equal length, we could use it in various problems such as in generating random na-Sequences or mapping random base-10 o-SSI sequences to corresponding na-Sequences. We set that sequence aside for later use in the following equation:

$$\Theta_{2na} == \psi_{na} \cdot \psi_{na} = \langle A, C, G, T, U, A, C, G, T, U \rangle \tag{10.27}$$

However, and also related, we might, also come to appreciate a related sequence, Θ_{palna} that is a palindrome of formed out of ψ_{na}, but still, with the same length as ψ_{10}.

$$\Theta_{palna} = \psi_{na} \cdot \neg(\psi_{na}) = \langle A, C, G, T, U, U, T, G, C, A \rangle \tag{10.28}$$

That said, and back to our **x-LGES** problem...

10.4.2 Revisiting Numeric Sequences

Before we proceed with actual na-Sequences, we shall first revisit hypothetical genome sequences expressed in numeric form, and then shall use the experience we garner to handle processing of actual na-Sequences later.

So, returning to that interesting **Hi-Fi o-SSI** sequence — $\Omega_{HiFinelle} = 8649137520$, if we are to convert it to an equivalent na-Sequence, we would obtain the results:

Transformation 20. $\Omega_{HiFinelle} = 8649137520 \xrightarrow{O_{na-dec-decode}(\cdot)} TCUUCTGAGA$

Which, if we wish to constrain it to just the DNA symbol set, would be as:

Transformation 21. $\Omega_{HiFinelle} = 8649137520 \xrightarrow{O_{na-dec-decode}(\cdot)} TCUUCTGAGA \xrightarrow{O_{mRNA-encode}(\cdot)} UCUUCUGAGA \xrightarrow{O_{DNA-encode}(\cdot)} TCTTCTGAGA$

And that transformation operated by chaining several transformers, some of which we have already encountered in earlier discussions, so that the final sequence is strictly DNA-encoded. Thus, our $\Omega_{HiFinelle}$ can be mapped from base-Ω into base-DNA and vice-versa as we have seen. These ideas can definitely then be used to process any other numeric sequence when there is need to map it to or treat is as a legitimate na-Sequence.

However, for the case of our gene expression system, we really don't care or need the na-Sequence expressions to begin with. And so, given we already have the numeral equivalent of $\Omega_{HiFinelle}$, we can directly advance to rendering the organism that might ensue from processing it via **x-LGES**. Note that, for this particular sequence, we had already dealt with its **IFA** prefix encoded in base-OZIN — see **Figure 10.3**. Thus, proceeding via a meaningful **transformatic-distillation** such as we did for the Euler Virus in **Transformation 18**

Transformation 22. $\Omega_{HiFinelle} = \langle 8,6,4,9,1,3,7,5,2,0\rangle \rightarrow \hbar(\overset{>}{\Omega}_{HiFinelle}) = \boldsymbol{816419111317151210 1} \rightarrow \overset{>}{\Omega}_{HiFinelle} = \langle 8,6,4,9,1,3,7,5,2,0\rangle \Longrightarrow \boxed{OZ(\Omega_{HiFinelle})} \approx \neg\langle 8649137520\rangle_{pf}$
$\Longrightarrow \boxed{OZ(\Omega_{HiFinelle})}^{*} = \langle 8649137520\rangle_{oz} \cdot \boxed{\langle 1350862479\rangle}_{pf} \cdot \langle 8649137520\rangle_{oz}$

Figure 10.12: An example of an actual organism from nature — an Earthworm, for which, like the x-LGES rendering of the sequence $\Omega_{HiFinelle}$, results in a genome expression for a creature that almost looks the same at the HEAD and ROOT. Image sourced from Wikimedia Commons[5].

Without actually wasting time and space rendering the visual equivalence of that organism, among interesting things to note from **Transformation 22** is that unlike in **Transformation 18**, we have applied the extended LGES algorithm — **Algorithm 7**, so that the final expression in that transformation contains the HEAD, BODY and ROOT of the organism. Also, and interesting to note, that despite this particular sequence having no exciting **IFA MSS** — since all the symbols in the original sequence have frequency 1 (which is why we see the organism's HEAD and ROOT looking exactly the same[21] *in numeric form*), though, because of the consequences of how the BODY component is generated, we see a potential mid-BODY-JOINT, that would definitely make the corresponding **FGE** interesting to look at.

10.4.3 x-LGES applied to named DNA Sequences

Now that we have attained sufficient experience with expressing pure numeric sequences, we can return to actual na-Sequences, and checkout some potentially enlightening scenarios.

First, we shall want to have a way to assign meaningful names to our sequences.

[21] We won't show many pictures here except perhaps **Figure 10.12**, but some kinds of **millipedes** and **centipedes** have this strange symmetrical anatomy that makes it difficult to distinguish its head from the tail. Earthworms too! Someone suggested the concept of the *Ouroboros* from esoteric folklore, that exhibits a kind of self-referential form, however, in my local folklores and vocabulary, we have a peculiar snake-like creature known as *namunungu* that exhibits this peculiar form too.

For example, the sequence from **Transformation 19**. Instead of just calling it Θ, we could adapt a name from its corresponding na-Sequence, and then use that to distinguish it from other na-Sequences we shall want to compare it against.

Without caring whether it actually has a standard or conventional name based on its actual composition as depicted, we could adopt the system such as picking the first letters of its contained codons in the order they appear in the sequence, and then assign it a name based on the resulting acronym. So, we for example know from **Table 9.1** originally associated with that sequence, that the corresponding codons (also see **Equation 9.8**) are as follows:

$$\Theta = \langle Met, Lys, Leu, Amber \rangle \tag{10.29}$$

So that we can just obtain a name for the sequence using the above suggested strategy as such:

Transformation 23. $\langle Met, Lys, Leu, Amber \rangle \rightarrow \langle M, L, L, A \rangle \rightarrow MLLA$

Thus, we might formally refer to our gene-program with a proper name as Θ_{MLLA} or just the **MLLA**-gene program — where no confusion might arise, we shall just refer to it as Θ_{ml}.

So, how would this program's corresponding sequence appear if it were expressed in the x-LGES?

We have its corresponding Ω-form expression in **Transformation 19**, so that:

$$\Theta_{ml} = \langle 1, 4, 3, 1, 1, 1, 4, 4, 1, 4, 1, 3 \rangle \tag{10.30}$$

From that, we can leverage the pre-rendering calculus we have already encountered in **Transformation 22**, and so that we apply **Algorithm 7** to Θ_{ml} as such:

TODO: fix the following derivation, also review all previous LGES and x-LGES diagrams as per highlighted tricky step in algorithm

Transformation 24. $\Theta_{ml} = \langle 1, 4, 3, 1, 1, 1, 4, 4, 1, 4, 1, 3 \rangle \rightarrow \hbar(\overset{>}{\Theta}_{ml}) = \boldsymbol{16432} \rightarrow \overset{>}{\Theta}_{ml} = \langle 1, 4, 3 \rangle \implies \boxed{OZ(\Theta_{ml})} \approx \neg\langle 1, 4, 3 \rangle_{pf} \implies \boxed{OZ(\Theta_{ml})}^{*} = \langle 1, 4, 3, 1, 1, 1, 4, 4, 1, 4, 1, 3 \rangle_{oz} \cdot \boxed{\langle 9, \rangle}_{pf} \cdot \langle 1, 4, 3 \rangle_{oz}$

Thus we conclude our treatment of TRANSFORMATICS in GENETICS.

Chapter 11

Conclusion

In this work, we have advanced our knowledge concerning the important and fundamental subject of genetics with new theory and practical ideas previously only held informally or totally unknown. We have explored several interesting ways by which the newly proposed mathematical field of inquiry, known as **TRANSFORMATICS**, might be applied in the sub-field of the biological sciences known as **GENETICS**.

Though much of the work was mathematical in nature, we have also had lots of useful theory proposed in the form of conceptual and especially philosophical ideas relating to how to understand or leverage genetics through the lens of a sequence analyst and/or a computer scientist.

In terms of **Key Metrics and Contributions** this single work has brought to surface:

1. **12 Working Definitions** were developed in relation to various concepts in analytical genetics, sequence analysis and sequence processing in general.
2. **4 Theorems** were developed and presented for the first time, spanning genetic sequences and then sequence complements in general.
3. **3 Laws** were distilled and presented for the first time:
   (a) **Law 1**: Concerning the nucleic acid identifier symbol set, ψ_{na}.
   (b) **Law 2**: Concerning Introns: Non-coding gene-programs.
   (c) **Law 3**: The **Identity Genome Sequence** Law.
4. **14 Transformers** were developed and presented for the first time, including the formal definition of the **Ribosome** as a **Protein Generator**.
5. **24 Transformations** were presented to help advance generic as well as particular computational solutions to many kinds of basic problems in sequence analysis, processing and one concerning gene/genome expression.
6. **5 Algorithms** were developed and presented for the first time, including:
   (a) **Algorithm 1**: The **Closest Sequence Algorithm**.

(b) **Algorithm 2**: The **tMCS** Algorithm for computing the Maximum Common Subsequence.

(c) **Algorithm 3**: The **Ω-to-OZIN Genetic Code Transcription Algorithm**.

(d) **Algorithm 4**: The **Ω-to-PLATONIC Form Genetic Code Expression Algorithm**.

(e) **Algorithm 5**: The **x-LGES Algorithm**.

(f) **Algorithm 6**: The **First Lu-Shuffle Algorithm**: lu_shuffler_a(Θ, k).

(g) **Algorithm 7**: The **Second Lu-Shuffle Algorithm**: lu_shuffler_b(Θ, k).

7. **3 Tabular Analyses** were presented, including **Table 6.1** that depicts how to perform comparative statistical analysis of DNA, RNA or any na-Sequence using some measures and methods from transformatics.
8. **2 Original Ciphers** — very rare and universally usable cryptographic keys were presented as part of the **Lu-Genome Sequence Expression System**, as well as proper justification for their practical relevance and sufficient historical background and example applications of both.
9. **3 Core Problems** in the matter/domain of **Genome Sequencing** were formally defined and their theoretical solutions also presented.
10. **More than 3 Source Code Examples**, some in the TEA language, and relating to conducting quick analyses of arbitrary na-Sequences via the commandline, and two (in **Appendix 8**), concerning how to randomize arbitrary sequences using Python.
11. And many other quantifiable and unquantifiable major and minor contributions to genetics, mathematics, computer science, philosophy and more.

In **Chapter 2** we have formally defined the DNA and RNA symbol sets, ψ_{DNA} and ψ_{RNA}. We have looked at the matter of the difference between the ordering of terms in these mathematically-oriented sets, relative to the common-place **A-T-C-G** ordering typical in biological and especially genetics literature. We have defined the more universal ψ_{na} that would help apply logic to both DNA and RNA sequences, and have generalized all such sequences as **na-Sequences**.

In **Chapter 3** we have looked at the idea of using n-grams such as the meaningful na-Sequence nucleobase 3-grams (technically known as **codons**) to abstract away various nuances of the flat-structure DNA or RNA sequence whose length might sometimes shoot into the millions or billions for realistic complete genome sequences. In **Chapter 9** we further these ideas to look at how to process na-Sequences at n-gram level such as in naming or identifying subsequences. We have also seen that such abstractions might help manage complexity and simplify formalisms or algorithms dealing with na-Sequences such in the formalization

of protein synthesis for natural or artificial automata via codon-processing ribosomes.

In **Chapter 4** we look at how we might leverage the basic concept of a symbol set to analyze arbitrary na-Sequences. We for example look at how to automatically generate special ordered symbol sets corresponding to individual na-Sequences under various bases — DNA, RNA, na- or even sometimes lexically meaningful AZ(ψ_{az}). We have seen how to define special symbol sets based on n-gram abstractions of na-Sequences, such as might help in logic with special na-Subsequences such as START and STOP codons.

In **Chapter 5** we then consider the interesting matter of **complement sequences** and **complement symbol sets**. We look at how to generate complements given a starting sequence or symbol set. We look into the interesting fact that complementing ψ_{DNA} helps us arrive at the common **A-T-C-G** ordering of terms such as in $\dot{\psi}_{DNA}$. Though we don't use it immediately, we see that we might also talk of orthogonal symbol sets and orthogonal symbol set identities relation to na-Sequences.

In **Chapter 6** we dive into how to leverage the **ADM** and **MSS** to perform non-trivial sequence analyses. We utilize four scenarios starting with basic scenarios between two individual sequences of potentially unequal lengths, all the way to analyzing entire collections or populations based on their representative na-Sequences. We look at how to leverage the **TEA** text-oriented GPL to conduct some of the essential analyses proposed, readily via the the commandline, and with sequences of any size stored in basic text files. We see that using the concept of the **modal sequence statistic**, whether applied to individual na-Sequences, collections of them or just sub-sequences of a longer DNA or RNA strand, could help us leverage statistical summary information about a sequence that might otherwise not be easily to glean or leverage. Especially, with previous work having shown us that traditional statistical measures and tests might sometimes fail to capture differences between actually different sequences, we further explore this idea as part of a detailed analysis of several DNA and RNA sequences across the major four different scenarios. We also learn that, where some obvious measures and techniques might fail to perform well or as expected, the careful use of the ADM, MSS and also newly defined **sequence characteristic** as well as **population characteristic** measures can offer the analyst new, better and indispensable information about one or more sequences while making comparisons or designing computational logic around the properties of sequences easier and repeatable. In this chapter too, particularly in **Section 6.6.1**, we develop an algorithm that might be utilized in bringing statistical artificial intelligence to the matter of sequence classification problems relating to labeled datasets. The closely related **Section 6.6.2** considers the matter of identifying

the **closest-sequence** from a collection of na-Sequences when given a particular query na-Sequence. Finally, **Section 6.7** considers the important matter of **optimal database storage of na-Sequences** as is typically found in real-world systems such as those that do whole-genome sequencing or gene-banks, etc. In particular, we develop a special sequencing index or identifier based on **Gödel Numbers** and our newly proposed sequence characteristic measure — this, so storage and sequence search and look-up problems can be conducted efficiently using information already in the sequences.

In **Chapter 7** we get into some deep mathematics around the critical problem of genome sequencing. We attack the matter by breaking it down into 3 major problems. **Problem G1** deals with the matter of how to compute the longest common subsequence given any two sequences. **Problem G2** deals with how, as happens during conventional genome sequencing, we have sections of two or more sequences that not exactly the same, but which must be reduced to some **consensus sequence** that is most true to all of them. And then in **Problem G3**, we consider how to arrive at the final best consensus sequence that also unifies all the disparate input sequences originally provided, into a single sequence that is then the genome sequence of the entity under analysis. We then formally define an abstract genome sequencing machine in **Section 7.4**, and culminate in an important result, the **Identity Genome Sequence Law** in **Section 7.5**.

We then turn our attention to the matter of how genetic code is translated into actual living organisms in **Chapter 9**. Focusing at the molecular level, we tackle the matter of the algorithm by which the most basic synthesis happens in a cell. Thus, **Section 9.1** presents the **protein synthesis algorithm**, and among important contributions of this work are the two diagrams — **Figure 9.2** and **Figure 9.3**, one a flow-chart for the algorithm, the other a **State Machine** for the Ribosome, the later also highlighting our concept of what the **Instruction Set** for a Ribosome could be — a useful concept for those planning on designing or implementing artificial ribosomes or bio-automata in simulation or kind.

Our final chapter, **Chapter 10**, looks at what results from actual processing of genome sequences or rather DNA code; the manifestation of some living thing, organism or some aspect of it as specified in the underlying genetic code. Thus, we explore **genome expression** formally. Via a thought-experiment, we propose a hypothetical genome-expression system — the **Lu-Genome Expression System** (LGES), and do break down the expression process into meaningful bits; **Section 10.1** deals with a visual encoding method that leverages a digit-cipher named **OZIN**, and this is used to express storage-level genetic code (in decimal/numeric form), it its **intermediate form** — equivalent to DNA-to-mRNA transcription in living organisms. The essential transcription rules are captured in **Algorithm 5**. Some examples of such transcription in LGES are shown in

Section 10.1.2. We finally turn attention to the final expression step — equivalent to mRNA-to-Protein/Function translation in nature, in **Section 10.2**, and also get to see yet another innovation in the form of a genetic-code-to-spatial-form encoding based on a cipher we call **Platonic Form Cipher** (**Section 10.2.1**). The essential algorithm for how to perform genome expression all the way from storage code into the final organism is depicted in **Algorithm 6**, and we close with some examples of complete, though hypothetical, fully-expressed genomes in **Section 10.2.3**.

So, with that introduction clearing up much of the basic genetics nomenclature and concepts that we shall use in the rest of this work, we then dive into the meat of our undertaking as such; **Chapter 2** shall introduce the use of terminology and notation from transformatics to describe facts about genome sequences at various levels of abstraction. We shall look at using sequence symbol sets, sequence abstraction using sub-sequence symbols that can later be re-transformed into flat sequences, the idea of sequence cardinality when applied to DNA sequences and more. Then in **Chapter 4** we shall deepen our discussion by considering how some of the measures from transformatics might be applied in genetic sequence analysis. We shall look into the ADM and PCR especially — the other for cases where any two sequences are of the same length and similar symbol sets, and the other for cases where these need not be the same across the sequences under analysis. Then in **Chapter 10** we shall take on the interesting matter for how, despite being merely a sequence of symbols, genetic code sequences actually can be likened to how a MSS can be used to specify how to reconstruct some other sequence. First, in **Chapter 9** we shall first look at an overview of how actual interpretation or execution (more conventionally just referred to as "gene translation") results in the manufacturing of proteins, and shall likewise look at some example actual gene translation in general and with a particular case. In **Section 9.2** we shall then dive into a more hands-on exploration of this matter, with a hypothetical genome system (the **Numeral-Gene Code**) that can allow us to not only creatively explore what genetic code is about, but which can also allow us to approach the rather complex matter of how DNA gets interpreted/translated into actual living tissue as well as a complete living organism. We shall introduce some two systems for how to decode our numero-gene DNA code into a kind of mRNA via the **Ozin-Transformer** and from ozin-gene code into actual tissue/proteins via the **Plato-Form Generator** that allows us to transcribe DNA into an orgnanism that has a predictable characteristic appearance and geometry, but also which still contains its essential genetic code just like normal living organisms do. Then we shall wrap-up in the **Conclusion**, looking at what we have accomplished, what remains to be done, and what the implications of this undertaking might be.

Appendix A

Origins of the OZIN Cipher

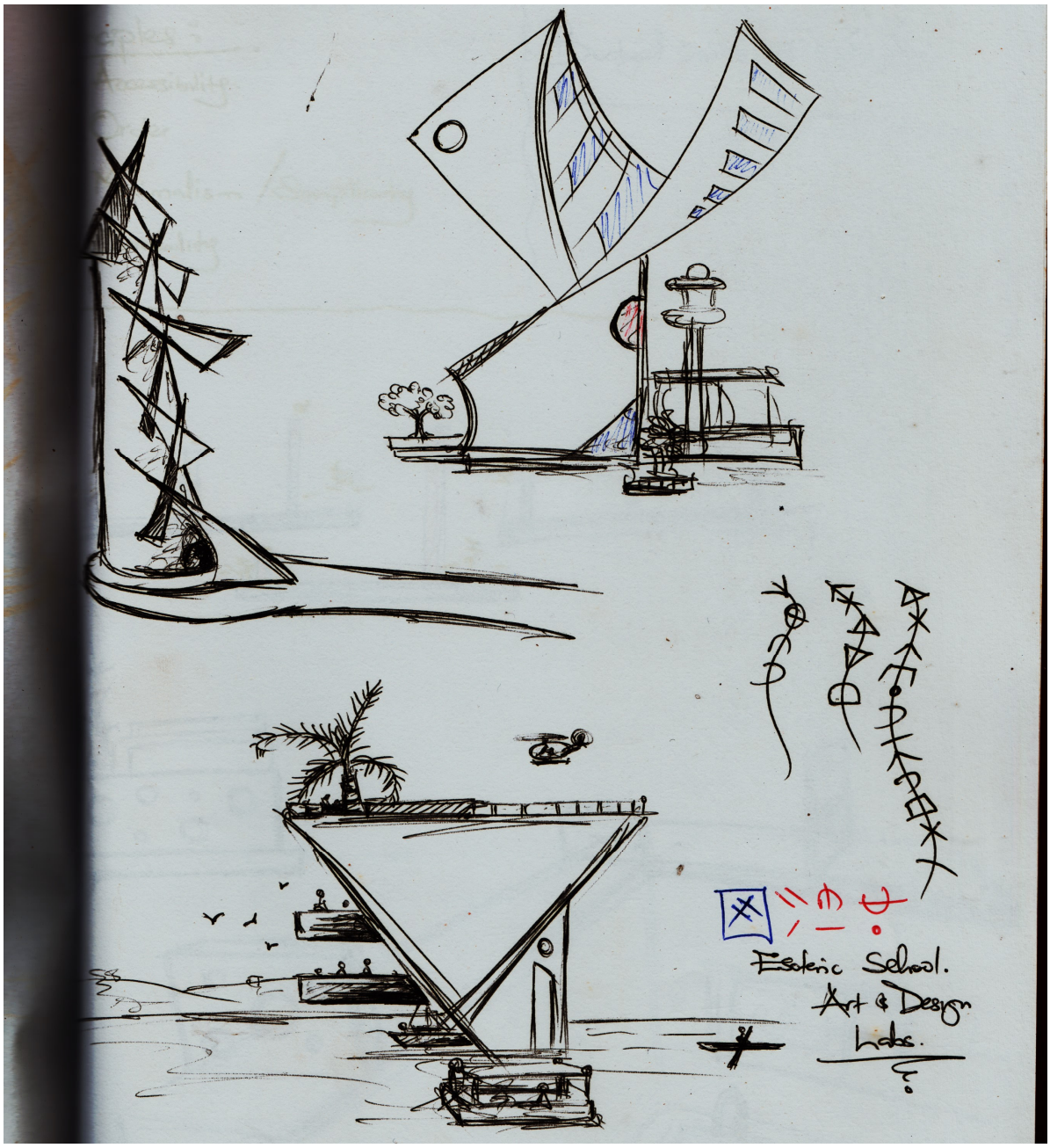


Figure A.1: An excerpt from architectural design notes by the author from early days of his home-based research laboratory. In this note, we also get to see an idea similar to how gene expressions are actually a chain of glyphs that, as in our **LGES** protocols, are depicted as structures chained to each other along a *backbone structure* symbolically depicted as a line (curved lines in this case). Copyrights due to author and Nuchwezi Research Labs.

For some, the **Ozin Cipher**[3] (see **Section 10.1**) might not seem that important or useful, but, perhaps because not everyone can immediately appreciate the fact that cryptography is such a dear part of expression in nature, that whoever is the Architect of Life, has applied it throughout nature in many creative ways — camouflage in reptiles and viruses, the occult languages of birds, etc. And these phenomena have indeed inspired many observant minds throughout the ages, to pick a leaf from nature and apply these ideas (in *alternative information expression*) in everything including mathematics, philosophy and yes... *magick*! Talking of which, for those interested in getting a deeper idea concerning how such a beautiful numeric symbol system — an alternative to the commonplace Arabic Numerals, came to be, one useful resource is the **CODE OGF**[55].

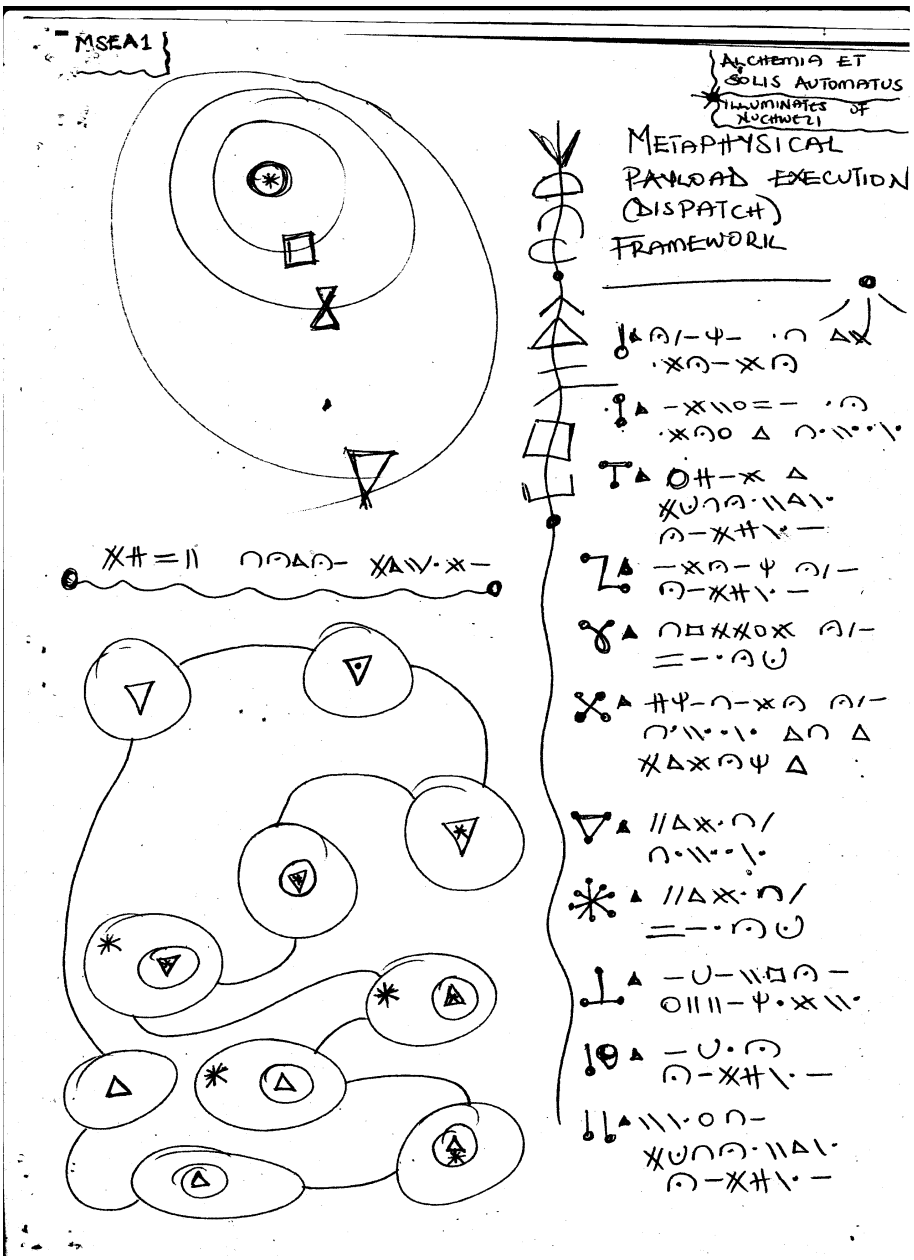


Figure A.2: The OZIN Cipher in use. An excerpt from the Church of Dance Eternal OGF

Figure A.2 is just an excerpt from that manuscript, perhaps not so clear because the original manuscript is handwritten, and this is just a scanned version. But hopefully it drives the point home. Also, as seen from that excerpt as well as a closely related excerpt — see **Figure A.3**, just like we might replace numbers with alternative symbols, we also see the Latin Alphabet replaced by an "alien-looking" hand, and this might definitely have its place in transformations of ψ_{az} sequences or their expression in other ways!

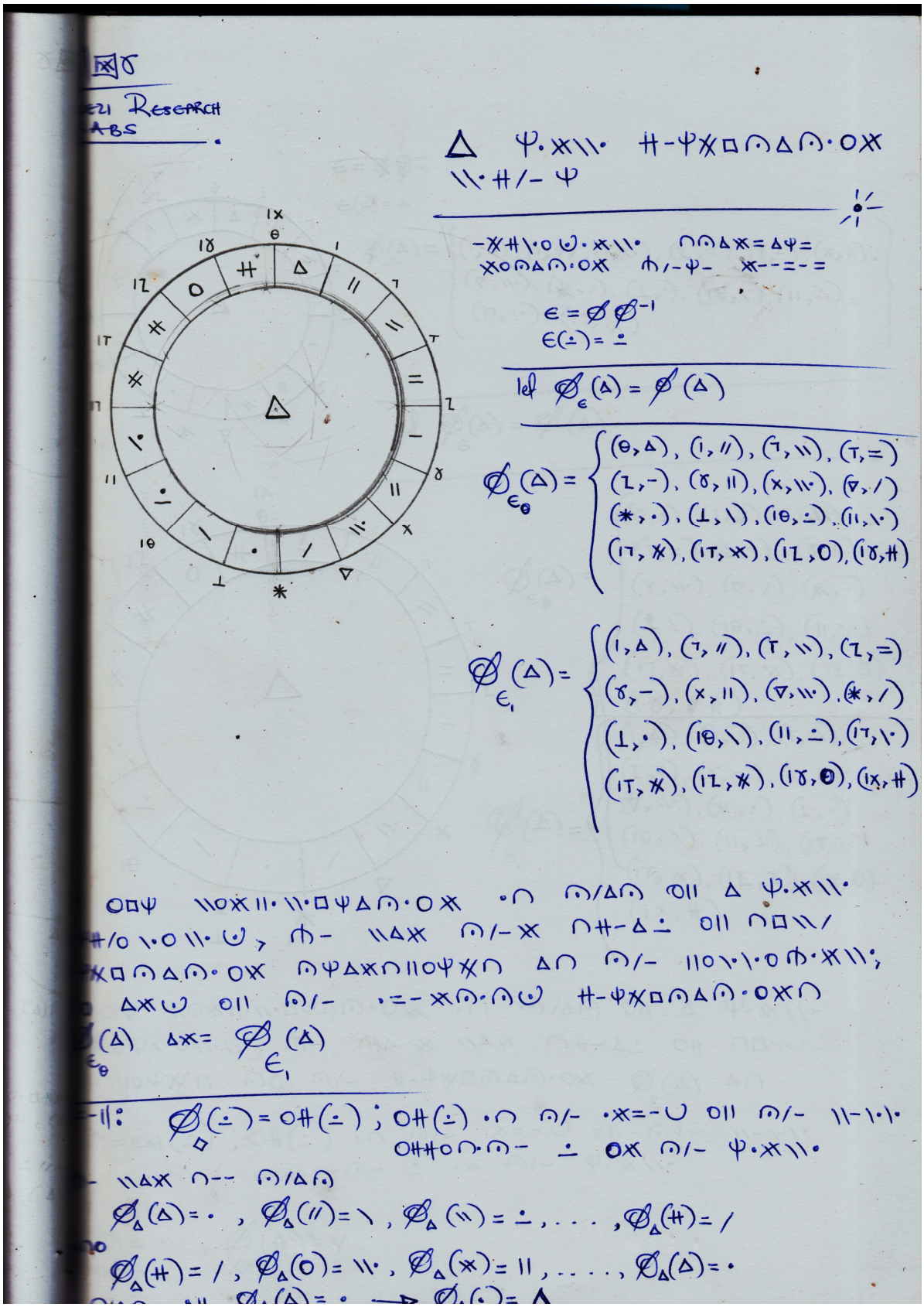


Figure A.3: The OZIN Cipher in use. An excerpt from author's research notes (circa 2019) on developing a cryptographic system leveraging a dial mechanism. Copyrights due to author and Nuchwezi Research Labs.

Appendix B

Computing Platonic-Form Encoding Keys (PFEKs), $\boxed{\psi_\Omega}_m$ for a Particular na-Sequence Θ or m

It is one of the tricky, but otherwise readily solvable aspects of the **Lu-Genome Expression System** (LGES), that when constructing the **BODY** aspect of a **Full Genome Expression** (FGE) corresponding to a particular sequence Θ, that we correctly pick or generate the corresponding platonic-form encoding key, $\boxed{\psi_\Omega}_m$, that corresponds to the sequence's characteristic, or rather, the **modal sequence**.

In particular, we use the concept of shuffling/anagrammatizing a sequence relative to some initial/specific input **partitioning index** k — in this case m, derived as such:

$$m = \text{ɯ}(\overset{>}{\Theta}) \tag{B.1}$$

So that, for the case of rendering **FGE** basing on some sequence Θ — possibly an na-Sequence, encoded (for **LGES**) using base-Ω, we use the **Second Lu-Shuffle Algorithm** (SLSA) — introduced in **Section 8.2**, and for which important example look-up-tables are shown in **Section 8.2.2**, to anagrammatize ψ_Ω **relative to** m, so as to arrive at a correct $\boxed{\psi_\Omega}_m$ that we then use to render the **FPA** for Θ. Essentially, we must compute the transform:

Transformation 25 (Generating the **Platonic-Form Encoding Key**).
$\psi_\Omega \xrightarrow{slsa(\psi_\Omega, m)} \boxed{\psi_\Omega}_m$

Where $slsa(\Theta, m)$ is equivalent to invoking `lu_shuffler_b`(Θ, m) as examples in **Listing 8.2** show.

Good enough, we not only have ready-to-use shuffled versions of $\psi_\Omega = \psi_{10} = \langle 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 \rangle$ as shown in **Figure 8.6**, but also, in case one wishes to manually compute or perhaps automatically generate the necessary key for a given m or *any* sequence, the Python program in **Listing 8.2** can be utilized for this purpose.

That said, basing on the algorithm that generated the mappings in textbfFigure 8.6, we have ready-to-use **Platonic-Form Encoding Keys** corresponding to particular values of m as shown in the following table.

PFEK Look-up Table by FLSA										
$m = \mu(\overrightarrow{\Theta})$ \ $I(\omega_i, \overrightarrow{\Theta})$	1	2	3	4	5	6	7	8	9	10
1	**9**	7	4	0	5	3	8	2	1	6
2	**9**	**7**	4	0	5	3	8	2	1	6
3	**8**	**6**	**3**	9	4	2	7	1	0	5
4	**8**	**6**	**3**	**9**	4	2	7	0	1	5
5	**5**	**3**	**0**	**6**	**1**	7	4	8	9	2
6	**5**	**1**	**2**	**6**	**3**	**7**	4	8	9	0
7	**0**	**8**	**9**	**1**	**5**	**2**	**6**	3	4	7
8	**3**	**8**	**9**	**4**	**2**	**5**	**6**	**0**	1	7
9	**9**	**1**	**2**	**3**	**8**	**4**	**5**	**6**	**7**	0
10	**9**	**5**	**6**	**7**	**8**	**0**	**1**	**2**	**3**	**4**

Table B.1: Platonic Form Encoding Keys (PFEK) mapped to corresponding values of $m = \mu(\overrightarrow{\Theta})$ using **FLSA**

And if one wishes to render their **PFA** — the platonic form aspect, of the **FGE** using the **SLSA**, then the corresponding table is as follows:

PFEK Look-up Table by SLSA										
$m = \text{ш}(\overset{>}{\Theta})$ \ $I(\omega_i, \overset{>}{\Theta})$	1	2	3	4	5	6	7	8	9	10
1	**4**	2	8	9	0	7	3	6	5	1
2	**4**	**2**	8	9	0	7	3	6	5	1
3	**7**	**5**	**3**	0	8	2	6	1	9	4
4	**1**	**6**	**4**	**7**	2	9	0	3	8	5
5	**2**	**1**	**0**	**6**	**8**	3	5	4	7	9
6	**7**	**1**	**2**	**4**	**6**	**8**	3	9	0	5
7	**5**	**9**	**2**	**3**	**7**	**0**	**4**	6	1	8
8	**9**	**1**	**5**	**4**	**3**	**2**	**7**	**8**	6	0
9	**5**	**6**	**3**	**9**	**0**	**4**	**2**	**7**	**1**	8
10	**6**	**3**	**8**	**2**	**1**	**9**	**7**	**4**	**0**	**5**

Table B.2: Platonic Form Encoding Keys (PFEK) mapped to corresponding values of $m = \text{ш}(\overset{>}{\Theta})$ using **SLSA**

B.0.1 Example of Resolving PFA Components, $\boxed{\omega_i^*}$

We shall not attempt to render a complete **FGE** here since that matter is already well catered for in the associated chapter and section — see **Section 10.2.3**, however, we shall attempt to set straight, the necessary procedure for:

1. Computing m for a given Ω-encoded sequence Θ:
   (a) Compute the **modal sequence statistic**, $\overset{>}{\Theta}$ — see **Definition 1** in [9].
   (b) Compute m from the cardinality of the modal sequence as: $m = \text{ш}(\overset{>}{\Theta})$.
2. Picking the correct **PFEK** for that m — i.e. $\boxed{\psi_{pf}}_m$:
   (a) If using **FLSA**, refer to **Table B.1**, and for any $m = k$, pick the row for which the first column has the value k.
   (b) Of course, for any $m = k$, the only useful values on that row for doing the **PFEK** operations are the **first** k **emboldened** values on the row corresponding to m. E.g. for $m = 2$ the necessary key is just $\langle 9, 7 \rangle$, not the entire row.
   (c) If using **SLSA**, lookup the **PFEK** using **Table B.2**, so that, for $m = 2$, we have $\boxed{\psi_{pf}}_2 = \langle 4, 2 \rangle$
3. Resolving the correct component from $\boxed{\psi_{pf}}_m$ corresponding to a particular element ω_i in the corresponding **IFA MSS**, $\overset{>}{\Theta}$.

(a) So, if our **IFA MSS** was just $\langle 0, 2, 1, 3, 6\rangle$ such as for Ω_3 from **Equation 10.4**, then taking note of the special case for any $\omega_i^* = 0$ — such as the first term in this case (see **Step#2(e)ii** in **Algorithm 6**), we merely map values of '0' in the **MSS** to just the **JOINT/GAP** element in **Figure 10.6**), for the other terms, such as $\omega_3^* = 3$, since the corresponding **PFEK** for $m = 5$ is $\boxed{\psi_{pf}}_5 = \langle 2, 1, 0, 6, 8\rangle$, then $\boxed{\omega_3^*} = 0$ as per the corresponding look-up mapping we see for such an **MSS**:

$$\begin{bmatrix} 0 & 2 & 1 & 3 & 6 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 2 & 1 & 0 & 6 & 8 \end{bmatrix}$$

And so, for the last term in that **MSS**, $\boxed{\omega_5^*} = 8$

4. How to render the final pf-encoded version of ω_i^*, i.e. $\boxed{\omega_i^*}$.

(a) So, now that you know the **PFEK** for your m, such as $\boxed{\psi_{pf}}_5 = \langle 2, 1, 0, 6, 8\rangle$ from above, and have the correct pf-term for a particular term in the **MSS** such as $\boxed{\omega_5^*} = 8$ in the example above, the corresponding pf-term is the platonic structure from **Figure 10.6** that corresponds to the index 8 — so that would be the **octagon** or rather **octa-gram** with nothing at the center. But also, given the term we are decoding is $\omega_5^* = 6$, then the overall structure would be as per configuration $8_{pf} + 8 \times 6_{oz}$: That is to say:

$$\boxed{\omega_5^*}_{pf} = 8_{pf} \otimes 6_{oz} = 8_{pf} + 8 \times 6_{oz} \tag{B.2}$$

So that the final rendered term for this element alone would be as shown:

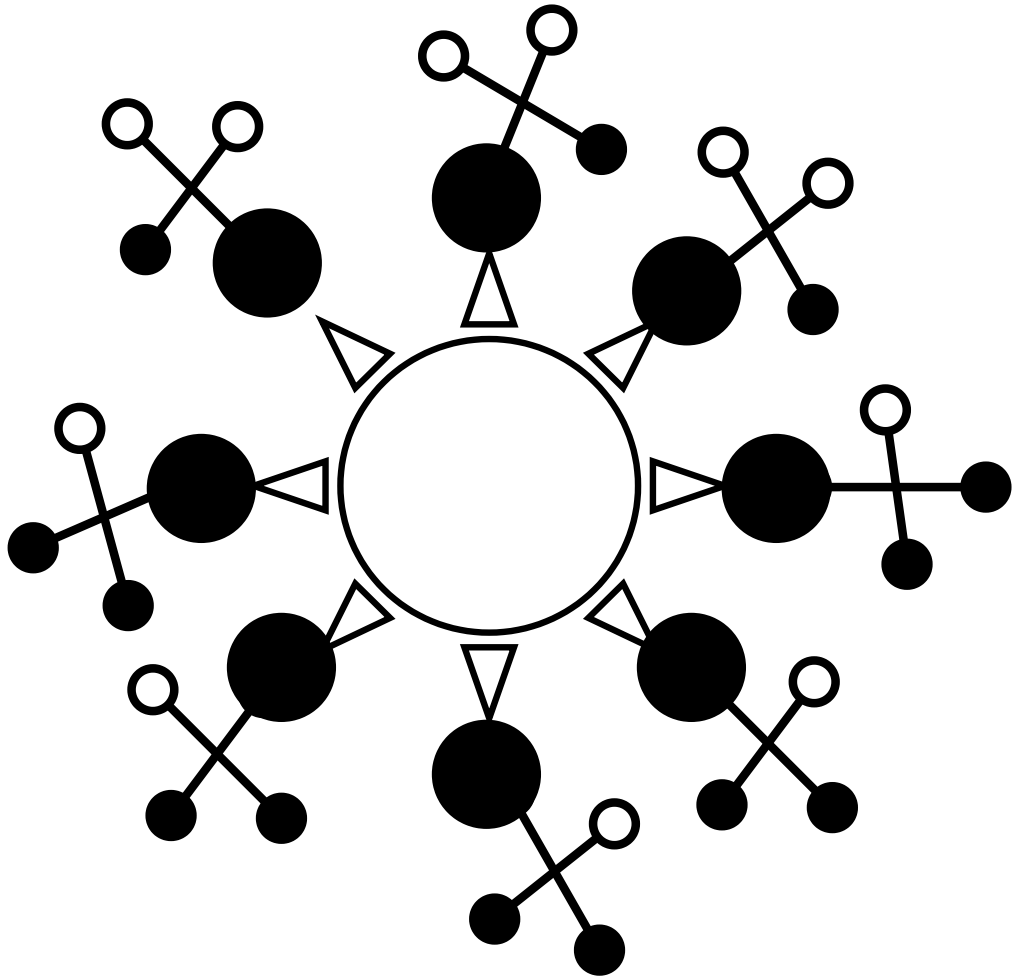


Figure B.1: An example of a single **PFA** component equivalent to the configuration $8_{pf} \otimes 6_{oz}$.

Of course, once one has the correct $\boxed{\omega_i^*}$, they can just attach it to the existing **FGE** chain-structure so as to construct the complete **BODY** aspect of a full genome expression for Θ.

Appendix C

Alternative[ORIGINAL] Ω-to-PLATONIC Form Genetic Code Expression Algorithm

While developing the theory for the **Lu-Genome Expression System**, and particularly while and after looking at the way the original concept of the algorithm for generating the BODY-components of the **FGE** was working — for example, it uses the **IFA MSS** as key to pick corresponding platonic form structures from $\neg(\psi_{pf})$ and not directly from ψ_{pf}, because the idea is that, the most frequent term in the **IFA MSS** should correspond to a *larger* or *more prominent* or *more complex* geometric form than the least significant terms. And thus, we shall see that, since natural genetic sequences (or rather, DNA-sequences) only span ψ_{DNA} which has only 4 distinct symbols, then we would never have a meaningful way to utilize the complete set of 10 distinct geometrical structures from ψ_{pf} — one interesting consequence of **Algorithm 8** (particularly **Equation C.7**) that performs the translation from **mRNA**-equivalent Ω-encoded **IFA MSS** to geometric pf-encoded **PFA**, using the $\neg(\psi_{pf})$, is that (**irrespective** of the organism or species we would look at) it seems as though the resulting **PFA** expressions **must** always **only** utilize the **first four** elements of $\neg(\psi_{pf})$ — $\langle 9, 8, 7, 6\rangle$, since all possible **IFA MSS** would only have utmost 4 distinct symbols in them. And if we had not used $\neg(\psi_{pf})$, still, a direct use of ψ_{pf} would only utilize the start of that symbol set if we map elements in ψ_{Ω} to those in ψ_{pf}, thus why we returned to the drawing board, and developed **Algorithm 6** instead.

Studying the resulting genome expressions visually, one would find that it seems like we always express the BODY of any **FGE** with some structure containing a **nucleus**[1][2]!

We shall keep that original algorithm here, for mostly study/comparison pur-

[1] Of course, the way the platonic forms corresponding to **9** and **7** were simplified to those of **8** (octagon-*chaosstar*-octagram) and **6**(hexagon-*hexagram*) respectively, with **the twist** that the **extra appendage point is placed at the center of the geometric structure instead of creating an extra vertex/appendage point**. It does have some good justification though — ***Occam's Razor*** *principle in a way, but also that, while observing nature, we typically find that* ***Simple is Better***. Nonetheless, the sensibilities of mapping any i in ψ_{10} or rather ψ_{Ω} to a platonic form corresponding to **the regular polygon with internal angle** corresponding to $\frac{360}{i}$ still makes sense for all 10 terms in ψ_{Ω}.

[2] The inverse would be **BODY** made of mostly gaps and/or joints, basic linear structures and triangles!

poses...

Algorithm 8 (The Ω**-to-PLATONIC Form Genetic Code Expression Algorithm**). *Assuming we have an Ω-base encoded genetic code sequence* Θ_Ω *of some length* $n > 0$.

1. *Initialize the expression with a transcription, by Using* ***Algorithm 5*** *to construct the* ***Intermediate Form Affix*** *(IFA) of* Θ_Ω *— essentially return* $OZ(\Theta_\Omega)$ *as the start of the* ***Final Genome Expression*** *(FGE)* $\boxed{OZ(\Theta_\Omega)}^*$ *— so that the final complete expressed(transcribed+translated) sequence shall be as:*

$$\boxed{OZ(\Theta_\Omega)}^* = OZ(\Theta_\Omega) \cdot \boxed{OZ(\Theta_\Omega)} \tag{C.1}$$

$\boxed{OZ(\Theta_\Omega)} : \mathbb{N} \times \psi_{pf}$ *is the suffix of expressed na-Sequence* Θ_Ω*, and is expressed in our genome expression system, as PLATONIC-Form encoded* ***modal sequence statistic*** *of* $OZ(\Theta_\Omega)$[3] *— the* ***PFA***[4]*, i.e.:*

$$\boxed{OZ(\Theta_\Omega)} = \boxed{\overset{>}{OZ}(\Theta_\Omega)}_{pf} \tag{C.2}$$

2. *So once we have the* ***IFA****,* $OZ(\Theta_\Omega)$*, proceed to use it to generate the rest of the sequence expression as follows:*

   (a) ***INITIALIZE*** *FGE,* $\boxed{OZ(\Theta_\Omega)}^*$ *as just:*

$$\boxed{OZ(\Theta_\Omega)}^* = \langle\langle OZ(\Theta_\Omega)\rangle\langle\rangle\rangle \tag{C.3}$$

   This ensures, we have the prefix affix, ***IFA****, as the contents of the sequence start of* ***FGE*** *extended by a* ***still empty subsequence*** *— in expression form, this shall merely be a backbone structure with* ***IFA*** *as its head, and*

[3]We render the **suffix of FGE** as $\overset{>}{OZ}(\Theta_\Omega)$ because it allows us to not express Θ_Ω naively at the genome or gene level, but also, so that, from an information-theoretic perspective, the completed genome expression is somewhat a statistical summary of what the genome sequence contains — the appearance of an animal or a plant or virus, depicting or expressing the summary of its DNA code. We do this then, by not merely doing a 1-to-1 translation of the intermediate Ozin-code expression $OZ(\Theta_\Omega)$, but instead computing its summary in form of $\overset{>}{OZ}(\Theta_\Omega)$, and then instead translating that into the final spatial/geometric forms as per ψ_{pf}. This shall then help complete the entire processing/synthesis of sequence Θ_Ω, or rather our genome expression.

[4]Platonic Form Aspect

a terminal that is ready to have more units appended to it, ***after a single GAP/JOINT extension to IFA****. So, it also means, at this juncture,* ***PFA*** *is just:*

$$\boxed{OZ(\Theta_\Omega)} = \langle\rangle \tag{C.4}$$

(b) ***COMPUTE*** *the* ***IFA MSS****,* $\overset{>}{OZ}(\Theta_\Omega)$

(c) ***FOREACH*** *symbol/term* ω_i^* *in* ***IFA MSS****,* $\overset{>}{OZ}(\Theta_\Omega)$*:*

i. ***USE*** *the translation mapping:* $\psi_\Omega \rightarrow \neg(\psi_{pf}) \rightarrow \psi_{pf}$ *as depicted in* ***Figure 10.6****, to translate* ω_i^* *into the equivalent* ***Platonic Form****,* $\boxed{\omega_i^*}$ *for the final code as such:*

ii. ***IF*** $\omega_i^* \equiv 0$*:Operating on the current* ***FGE****:*

A. Append/Insert a JOINT onto ***FGE****,* $\boxed{OZ(\Theta_\Omega)}^*$

$$\boxed{OZ(\Theta_\Omega)}^* = \boxed{OZ(\Theta_\Omega)}^* \cdot \boxed{\omega_i^*}_{joint} \tag{C.5}$$

And equivalently:

$$\boxed{OZ(\Theta_\Omega)} = \boxed{OZ(\Theta_\Omega)} \cdot \boxed{\omega_i^*}_{joint} \tag{C.6}$$

B. ***PROCEED*** *to the next symbol in* ***IFA MSS****,* $\overset{>}{OZ}(\Theta_\Omega)$ *(loop from* ***Step#2(c))****.*

iii. ***ELSE****:* ***resolve*** *the correct* $\boxed{\omega_i^*}$ *for* ω_i^* *by using the relation:*

$$\boxed{\omega_i^*} = \rho_j + j \times \omega_i^* : j = I(\rho_j, \neg(\psi_{pl})) = I(\omega_i^*, \psi_\Omega) \tag{C.7}$$

A. ***PLACE*** *that encoded version,* $\boxed{\omega_i^*}$*, along the existing* ***FGE*** *structure at the next vacant position* $(\mathfrak{u}(\Theta_\Omega)+i)$*. Expressible with another version of* ***Equation 10.13*** *as:*

$$\boxed{OZ(\Theta_\Omega)}^* = \boxed{OZ(\Theta_\Omega)}^* \cdot \boxed{\omega_i^*} \tag{C.8}$$

And equivalently:

$$\boxed{OZ(\Theta_\Omega)} = \boxed{OZ(\Theta_\Omega)} \cdot \boxed{\omega_i^*} \tag{C.9}$$

Where the special term, $\boxed{\omega_i^*}$*, as per* ***Equation 10.15****, is basically the PLATONIC Form corresponding to position* j *in* $\neg(\psi_{pf})$ *— a* ***complement/inverted symbol set****, thus one of the final form platonic geometry structures, but also with the intermediate form* ω_i^* *expressed* j *times as an attachment to* ρ_j*, such that:*

$$0 \leq (j = (\text{ɯ}(\psi_{pf}) - i)) \leq (\text{ɯ}(\psi_{pf}) = \text{ɯ}(\psi_\Omega)) \geq \text{ɯ}(OZ\overset{>}{(}\Theta_\Omega)) \quad \text{(C.10)}$$

as depicted in example ***Figure 10.7****, with each expression of* ω_i^* *attached to one of the* j ***appendage spots*** *on the shape for* $\boxed{\omega_i^*}$*.*

B. ***PROCEED*** *to the next symbol in* ***IFA MSS****,* $OZ\overset{>}{(}\Theta_\Omega)$ *(loop from* ***Step#2(c))****.*

3. ***TRUNCATE*** *the final-structure after the final term in* ***FGE****.*
4. ***RETURN FGE****,* $\boxed{OZ(\Theta_\Omega)}^*$ *as the final transcribed and translated version of* Θ_Ω*. The culmination of our conceptual genome expression.*

□

NOTE:

Besides the introductory notes about this alternative algorithm given at the start of this chapter, it might be worth the effort, looking at what kinds of expressions the algorithm can produce so one has a basic means to weigh it against the version we prefer/chose (**Algorithm 6** in **Section 10.2.2**). Thus, without delving much into how they were generated, we shall [re-]iterate some of the highlights of how **Algorithm 8** would render various sequences under the **Lu-Genome Expression System**.

IMPORTANT NOTE: Realize that **Step#2(c)iii(A)** of **Algorithm 8** processes/operates on the **IFA MSS**, $OZ\overset{>}{(}\Theta_\Omega)$, and not the **IFA** directly. This, so that, first of all, we don't process multiple times each term or symbol from the **IFA** — which is a direct translation of the genetic code, in the **PFA**. And then also, the way the translation works, is that if $\omega_x^* > \omega_y^*$ or rather that $|\omega_x^*| > |\omega_y^*|$, then ω_x^* shall correspond to a term in $\neg(\psi_{pf})$ that is simpler[5] in form than ω_y^* — higher significance being assigned to earlier-occurrence in the underlying symbol set ψ_Ω.

Good enough, we have already treated of sequence complements and inverted look-ups in **Section 5.1**, for this case, the corresponding mapping from terms (or

[5]The idea was somewhat to assign more complex forms to terms in the associated symbol set that occur earlier than others since it somewhat depicts their relative higher significance — a concept similar to how we would attempt to regenerate a sequence from its modal sequence statistic without say consulting its sequence characteristic — an inversion of and adaptation of the **string chart** concept first presented in the **LOMT-1 Transformer** (**Transformer 6** in [9]).

rather, magnitudes) in $OZ(\overset{>}{\Theta}_\Omega)$ to indices in $\neg(\psi_{pf})$ is as below — also stressing the fact that $0 \leq |\omega_i^*| = |\omega_i| < \mathscr{u}(\psi_{pf})$ for all $\omega_i \in \Theta_\Omega \quad \wedge \quad \omega_i^* \in \overset{>}{\Theta}_\Omega$.

$$\begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \end{bmatrix}$$

Looking at some example genome expressions rendered using **Algorithm 8** shall help most reviewers to appreciate whether or not it is better or worse than **Algorithm 6**, and whether or not it is better at representing how natural genome expression occurs as seen in our inspiration case — see **Figure 1.1**.

C.0.1 Examples of Complete Lu-Genome Expressions

We get to see the first example of the output of processing some na-Sequence expressed originally in base-Ω such as for Ω_3 from **Equation 10.4**, in its final expression form as in **Figure C.1**[48] — more importantly, since we have already seen its **IFA** (the **FGE** prefix) in **Figure 10.4**, we note that we essentially see the update as the result of expressing the entire **FGE** via the **IFA MSS** equivalent to:

$$\overset{>}{\Omega}_3 = \langle 0, 2, 1, 3, 6 \rangle \implies \boxed{OZ(\Omega_3)} \approx \langle GAP, 7, 8, 6, 3 \rangle \tag{C.11}$$

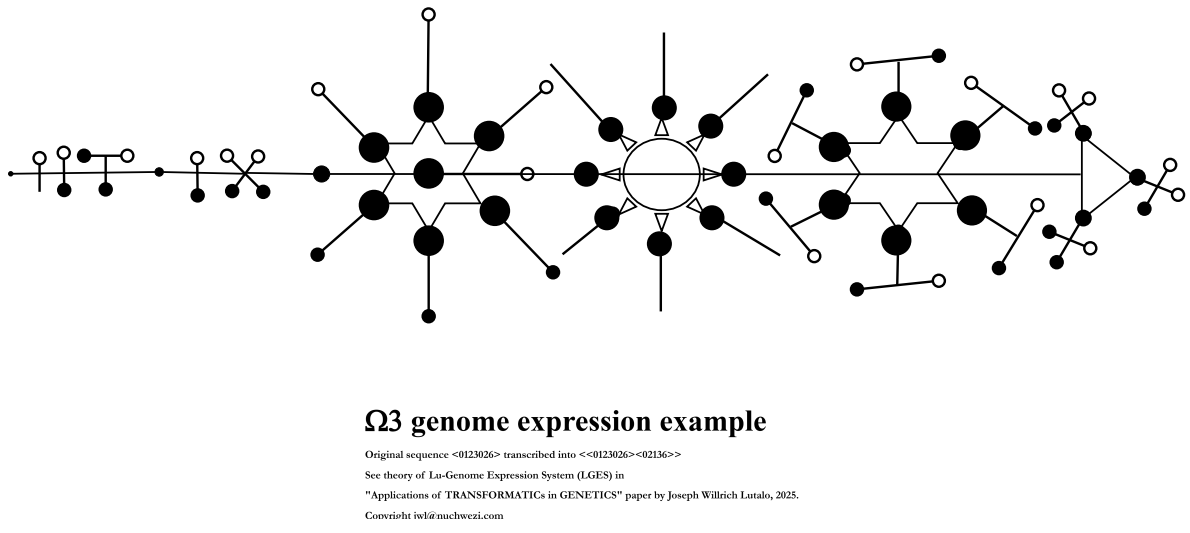


Figure C.1: The equivalent LGES complete genome expression of the Ω_3 genome sequence (from **Equation 10.4**)

Though we won't delve into exploring all kinds of possible *interesting* scenarios with regards to the **LGES**, we can look at one other example and then move on.

From the example hypothetical genome sequences we have encountered, perhaps let us revisit **Equation 10.2** and see what the **Euler Virus** might look like when fully expressed in our system.

So, note that we already did the first-leg of **Algorithm 8**, and have seen what the **FGE** prefix — the **IFA**, would look like in **Figure 10.2** So, like we did for Ω_3 in the last exploration, let us start by looking at what the numeral-equivalence of the processed Ω_{veuler} would be like. For brevity, we can sum up the entire production/transformation that leads to the complete **FGE** with correct prefix and suffix, and thus $\boxed{OZ(\Omega_{veuler})}^*$ as:

Transformation 26. $\Omega_{veuler} = \langle 2,7,1,8,2,8,1,8\rangle \rightarrow \hbar(\overset{>}{\Omega}_{veuler}) = \boldsymbol{83221271} \rightarrow \overset{>}{\Omega}_{veuler} = \langle 8,2,1,7\rangle \implies \boxed{OZ(\Omega_{veuler})} \approx \neg\langle 8,2,1,7\rangle_{pf} \implies \boxed{OZ(\Omega_{veuler})}^* = \langle 2,7,1,8,2,8,1,8\rangle_{oz} \cdot \boxed{\langle 1,7,8,2\rangle}_{pf}$

Note that in that derivation depicted in **Transformation 26**, one of the most tricky parts is computing that FGE suffix, $\boxed{OZ(\Omega_{veuler})}$, the **PFA** — which is formalized/defined by *especially*[6] **Equation C.7**. So, once that is done, and we have our final numeral-**FGE** as such:

$$\boxed{OZ(\Omega_{veuler})}^* = \langle 2,7,1,8,2,8,1,8\rangle_{oz} \cdot \boxed{\langle 1,7,8,2\rangle}_{pf} \tag{C.12}$$

Of course, concerning **Equation C.12**, note that, or rather, keep in mind that concerning the **PFA** component, $\boxed{OZ(\Omega_{veuler})}$, the corresponding associated sequence in numeral form is the **IFA MSS**, $OZ(\overset{>}{\Omega}_{veuler}) = \overset{>}{\Omega}_{veuler} = \langle 8,2,1,7\rangle$. And thus, using necessary transcription and translation ciphers as well as the multiplication rules required by **Equation C.7** when treating of the **PFA**, we then generate the final genome expression for the **Euler Virus** under the **LGES** that looks as in **Figure C.2**.

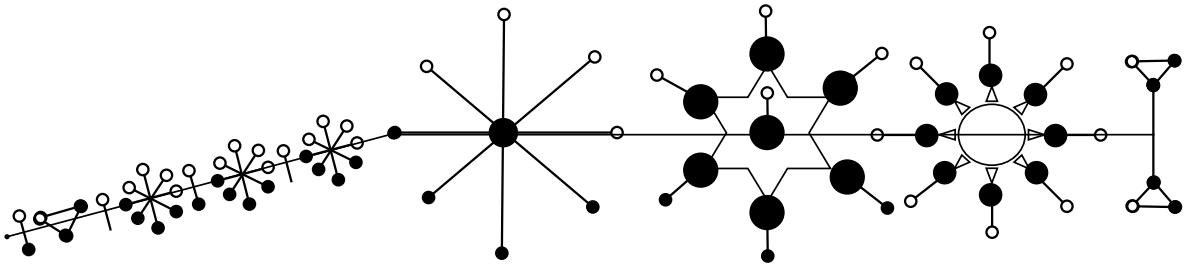


Figure C.2: The equivalent LGES complete genome expression of the Ω_{veuler} (Euler Virus) genome sequence (from **Equation C.12**)

[6] **Equation C.6** is what covers the special symbol '0', that is treated as a JOINT/GAP in the **PFA** and as STOP/GAP in **IFA**.

Now, concerning that rendering of the genome sequence expressed in **Equation C.12**, it might be worth noting that unlike the earlier examples of **LGES** expressions, we have taken the liberty to exploit that special joint that kick-starts the **PFA**, or rather, which separates the **IFA** from the **PFA** in a **FGE** expression. So, just like an organism with a physical joint might sometimes appear as angled or curved, so we have chosen to depict the Ω_{veuler} with its one joint[7] well utilized. However, and important to note; it is not just joints that might appear different across different renderings of the same basis genome sequence. So, in **Figure C.3**, we see another variation of the same sequence, with readily noticeable differences being the IFA-PFA joint in its neutral position, and then the second node in the PFA (corresponding to $7_{pf} \otimes 2_{oz}$ or rather $7_{pf} + 7 \times 2_{oz}$) with the 2_{oz} appendages depicted mostly aligned horizontally.

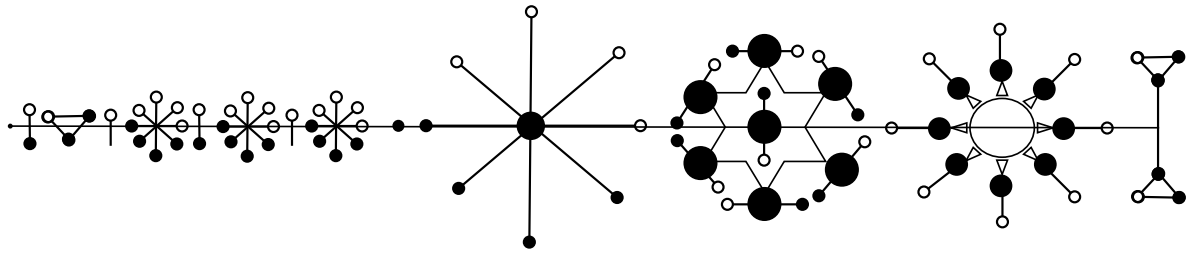


Figure C.3: The **alternative rendering** of the same sequence as Ω_{veuler}.

Concerning **Figure C.3**, the alternative rendering of the same sequence as Ω_{veuler} first depicted in **Figure C.2**, we see that underlying the variations is how the *degrees of freedom* in rendering the genome have somewhat been exploited. Notice how the joint and appendages are placed or oriented about the platonic forms that mostly remain unchanged across the two renderings — except perhaps in scale. And talking of scaling nodes, in the future, we might want to explore whether in such a system as **LGES** in which the expression is sensitive to measures such as the **IFA MSS**, whether it might make sense to scale each subsequent pf-node in the **PFA** based on its position within the sequence? Though we didn't have such a rule included in **Algorithm 8**, and yet, our example renderings here have somewhat taken advantage of that plausible logic.

That said, note that we started this work with a photograph of a pineapple ready to harvest (see **Figure 1.1**), and though the associated discussion[8] by the author did bring up the matter of how it might inspire a genome expression system in which the prefix is the actual DNA that renders the rest of the organism (fruit, plant-base and roots in this case of the pineapple), it is left as an exercise to the reader to decide whether example genome sequence expressions/renderings such

[7]Definitely, and interesting to explore, a genome sequence with multiple JOINT points in it might be how an organism with say appendages such as legs, hands and a head that can move independently of the core/thorax of the body is possibly arrived at. The interested student should go ahead and explore more non-trivial Ω-base sequences just to see what's possible.

[8]Checkout video/mini-lecture on the matter at author's YouTube: `https://youtube.com/shorts/yTJzBEpw7Z4`

as in **Figure C.2** depict this pineapple-inspired philosophy.

Applying TRANSFORMATICS

“A woman is a string that is a replicating enclosure. You invest a substring of your string in her for a while, and then she transforms it into a higher string which she eventually kicks out into the world. The woman is a very special transformer, the man mostly a generator. That’s most of natural biology expressed using TRANSFORMATICS.

Another way to look at it; Tukaruga ha musaana, twaija omunsi, yatuzaara.. kyakweta okutufoora. Baitu emara netubinga kugenda omumwanya kugarukayo... Kikuzooka niho KITARA.

So, it’s perhaps merely humbling to see how a basic mathematics originally meant to explain artificial intelligence and abstract machines, automatons, can likewise explain or express the creative, dynamic science of biological systems as well as the queer idea that life is an investment by the sun into planetary ecosystems, and which can later escape their closures to further express it in distant realms such as in remote galaxies and space-times!”

— a reflection about the new mathematical field of TRANSFORMATICS
as applied to explaining various natural and artificial phenomenon.

Foundation Paper: https://bit.ly/transformatics101

fut. prof. J. Willrich
(currently at Nuchwezi Research)

Bibliography

[1] Wikipedia contributors. Base pair. https://en.wikipedia.org/wiki/Base_pair, 2025. Accessed August 2, 2025. Includes diagram of DNA double helix showing A–T and C–G base pairing.

[2] GeeksforGeeks. Diagram of protein synthesis. https://www.geeksforgeeks.org/biology/protein-synthesis-diagram/, 2024. Accessed July 29, 2025. Shows transcription, translation, and ribosome-mediated protein synthesis.

[3] Joseph Willrich Lutalo. The ozin cipher. https://figshare.com/articles/onlineresource/The_OZIN_Cipher/29867474/1, Aug 2025.

[4] Joseph Willrich Lutalo. The plato cipher. https://figshare.com/articles/onlineresource/The_PLATO_Cipher/29867843/1, Aug 2025.

[5] Fir0002 / Flagstaffotos. Earthworm.jpg, 2007. Image taken in Swifts Creek, Victoria. Licensed under CC BY-NC and GFDL 1.2. Attribution required to Fir0002/Flagstaffotos.

[6] Joseph Willrich Lutalo. *SECTION 5: THE AFTERMATH: THE NEW CHWEZI*. I*POW, 2023. The official, free-access electronic edition is accessible via: https://t.me/ipowriters/26.

[7] Joseph Willrich Lutalo. Statement of purpose (university of oxford). https://figshare.com/articles/onlineresource/Statement_of_Purpose_University_of_Oxford_/29885561/1, Aug 2025.

[8] Joseph Willrich Lutalo. PROPOSED DOCTORAL THESIS: A Theory of Entropy Sources & Random Number Generators: DPhil Computer Science, Oxford University, Joseph Willrich Lutalo MSc. BSc. alumni Makerere University. 6 2025. Accessible via https://figshare.com/articles/online_resource/PROPOSED_DOCTORAL_THESIS_A_Theory_of_Entropy_Sources_Random_Number_Generators_DPhil_Computer_Science_Oxford_University_Joseph_Willrich_Lutalo_MSc_BSc_alumni_Makerere_University/29209706.

[9] Joseph Willrich Lutalo. **The Theory of Sequence Transformers & their Statistics**: The 3 information sequence transformer families (anagrammatizers, protractors, compressors) and 4 new and relevant statistical measures

applicable to them: Anagram distance, modal sequence statistic, transformation compression ratio and piecemeal compression ratio. *Academia.edu.*, 2025. https://doi.org/10.6084/m9.figshare.29505824.v3.

[10] mcnemesis. cli_tttt: Command line interface for tttt. https://github.com/mcnemesis/cli_tttt/, 2024. Accessed: 4th Aug, 2025.

[11] Carl Sagan. *Cosmos*. Book Club Associates, London, UK, 1981. Special edition for **Book Club Associates**.

[12] Richard L. Gregory and Oliver L. Zangwill, editors. *The Oxford Companion to the Mind*. Oxford University Press, Oxford, UK, 1987. Available online at https://archive.org/details/oxfordcompaniont00greg.

[13] BioExplorer.net. Do bacteria have nucleus? https://www.bioexplorer.net/do-bacteria-have-nucleus.html/, 2025.

[14] Seungho Kang, Alexander K Tice, Frederick W Spiegel, Jeffrey D Silberman, Tomáš Pánek, Ivan Čepička, Martin Kostka, Anush Kosakyan, Daniel M C Alcântara, Andrew J Roger, et al. Between a pod and a hard test: The deep evolution of amoebae. *Molecular Biology and Evolution*, 34(9):2258–2270, 2017. https://academic.oup.com/mbe/article/34/9/2258/3827454.

[15] OpenStax Biology. Viruses - biology libretexts. https://bio.libretexts.org/Bookshelves/Introductory_and_General_Biology/General_Biology_1e_(OpenStax)/5:_Biological_Diversity/21:_Viruses, 2025.

[16] Grady Venville and Jenny Donovan. Analogies for life: a subjective view of analogies and metaphors used to teach about genes and dna. *Teaching Science*, 52(1):18–22, 2006. Available at: https://research-repository.uwa.edu.au/en/publications/analogies-for-life-a-subjective-view-of-analogies-and-metaphors-u.

[17] University of Nebraska–Lincoln. Complementary, antiparallel dna strands — dna and chromosome structure. https://passel2.unl.edu/view/lesson/6f214d098527/4, n.d. Accessed July 29, 2025.

[18] Genomics Education Programme. Where does our genome come from? https://www.genomicseducation.hee.nhs.uk/education/core-concepts/where-does-our-genome-come-from/, 2025. Accessed July 29, 2025. Explains how sperm and egg each contribute half the genome, forming a unique combination in the zygote.

[19] OpenStax Biology. Gametogenesis (spermatogenesis and oogenesis). https://bio.libretexts.org/Bookshelves/Introductory_and_General_Biology/General_Biology_1e_(OpenStax)/43:_Animal_Reproduction_and_Development/43.3C:_Gametogenesis_(Spermatogenesis_and_Oogenesis), 2025. Accessed July 29, 2025.

[20] University of Leicester. The cell cycle, mitosis and meiosis for higher education. https://le.ac.uk/vgec/topics/cell-cycle/the-cell-cycle-higher-education, n.d. Accessed July 29, 2025.

[21] Joseph Willrich Lutalo. Concerning a special summation that preserves the base-10 orthogonal symbol set identity in both addends and the sum. *Academia*, 2025. *Accessible via* https://www.academia.edu/download/122499576/The_Symbol_Set_Identity_paper_Joseph_Willrich_Lutalo_25APR2025.pdf.

[22] Joseph Willrich Lutalo. Tea taz - transforming executable alphabet a: to z: Command space specification. https://doi.org/10.6084/m9.figshare.26661328, 2024.

[23] Shinichi Mochizuki. The geometry of frobenioids i: The general theory. https://www.kurims.kyoto-u.ac.jp/~motizuki/The-Geometry-of-Frobenioids-I.pdf, 2008. Accessed August 2, 2025. Introduces Frobenioids as symbolic categorical structures encoding arithmetic transformations.

[24] Microsoft Copilot. Clarifying discussions on dna sequence facts and genetics in general during drafting manuscript. AI-generated insights via Copilot discussion, 2025. Personal communication, July 2025.

[25] Wikipedia contributors. Nucleic acid sequence. https://en.wikipedia.org/wiki/Nucleic_acid_sequence, 2025. Accessed July 2025.

[26] Joseph Willrich Lutalo. A general theory of number cardinality. *Academia.edu*, Jan 2024. *Accessible via* https://www.academia.edu/43197243/A_General_Theory_of_Number_Cardinality.

[27] Nature Education. The four bases – atcg. https://www.nature.com/scitable/content/the-four-bases-atcg-6491969/, n.d. Accessed July 2025.

[28] Regina Bailey. Genetic code and rna codon table. https://www.thoughtco.com/genetic-code-373449, 2019. Accessed July 2025. Explains RNA nucleotide composition and codon structure.

[29] National Center for Biotechnology Information (NCBI). Refseq: Ins homo sapiens insulin [nm_000207.2]. https://databases.lovd.nl/shared/refseq/INS_NM_000207.2_codingDNA.html, 2020. Accessed July 31, 2025.

[30] J. Craig Venter et al. The sequence of the human genome. *Science*, 291(5507):1304–1351, 2001. Available at: https://www.science.org/doi/pdf/10.1126/science.1058040.

[31] S. Anderson, A. T. Bankier, B. G. Barrell, et al. Sequence and organization of the human mitochondrial genome. *Nature*, 290:457–465, 1981. Available at: https://www.nature.com/articles/290457a0.pdf.

[32] Joseph Willrich Lutalo. Introducing the anagram distance statistic, Ã, a quantifier of lexical proximity for base-10 o-ssi and any arbitrary length ordered sequences, its relevance and proposed applications in computer science, engineering and mathematical statistics. *Academia.edu*, 2025. *Accessible via* `https://doi.org/10.6084/m9.figshare.29402363`.

[33] Joseph Willrich Lutalo. Concerning debugging in tea and the tea software operating environment. *Academia.edu*, 2025.

[34] Joseph Willrich Lutalo. Philosophical and mathematical foundations of a number generating system: The lu-number system. *Academia.edu*, 2025. *Accessible via* `https://doi.org/10.6084/m9.figshare.29262749`.

[35] Wikipedia contributors. Dna and rna codon tables. `https://en.wikipedia.org/wiki/DNA_and_RNA_codon_tables`, 2025. Accessed August 2, 2025. Provides codon-to-amino acid mappings and genetic code/name tables.

[36] Shusei Sato, Yasukazu Nakamura, Takakazu Kaneko, Erika Asamizu, and Satoshi Tabata. Complete structure of the chloroplast genome of arabidopsis thaliana. *DNA Research*, 6(5):283–290, 1999.

[37] Valerie Illingworth, editor. *Oxford Dictionary of Computing*. Oxford University Press, Oxford, UK, 4 edition, 1996. Available at `https://openlibrary.org/books/OL412451M`.

[38] Joseph Willrich Lutalo. Tea research: Tea on the web a high-level web software operating environment specification for the tea programming language: Web tea architecture. `https://doi.org/10.6084/M9.FIGSHARE.29591687`, n.d. figshare.

[39] Kurt Gödel. Über formal unentscheidbare sätze der principia mathematica und verwandter systeme i. *Monatshefte für Mathematik und Physik*, 38:173–198, 1931.

[40] Kurt Gödel. On formally undecidable propositions of principia mathematica and related systems i. In Solomon Feferman, John W. Dawson Jr., Stephen C. Kleene, Gregory H. Moore, Robert M. Solovay, and Jean van Heijenoort, editors, *Collected Works, Volume I: Publications 1929–1936*, pages 144–195. Oxford University Press, 1986. English translation of Gödel's 1931 paper, accessible via: `https://archive.org/details/collectedworks0001gode`.

[41] Teresa Przytycka. Lecture 10: Whole genome sequencing and analysis. `https://www.ncbi.xyz/CBBresearch/Przytycka/download/lectures/PCB_Lect10_Whole_Genome.pdf`, 2025. Accessed August 2, 2025. Lecture slides from Introduction to Computational Biology.

[42] Susha Cheriyedath. Start and stop codons. `https://www.news-medical.net/life-sciences/START-and-STOP-Codons.aspx`, 2019. Accessed July 29, 2025. Describes canonical and alternative start codons across organisms.

[43] Joseph Willrich Lutalo. Software language engineering-text processing language design, implementation, evaluation methods. *Preprints*, 2024. *Accessible via* https://www.preprints.org/frontend/manuscript/3903e4cd075074a7005cb705a5ef26c5/download_pub.

[44] Joseph Willrich Lutalo. Applying transformatics: Sequence generators. https://doi.org/10.6084/M9.FIGSHARE.29654645, 2025. FigShare.

[45] Joseph Willrich Lutalo. Pragmatic computational mysticism. https://doi.org/10.6084/M9.FIGSHARE.27187071, 2024. figshare.

[46] Joseph Willrich Lutalo. Introducing zha, a real q-agi. *FigShare*, 2025. *Accessible via* https://doi.org/10.6084/M9.FIGSHARE.29049794.

[47] Joseph Willrich Lutalo. Unraveling mysteries of the zha q-agi chatbot: an interview by icc, of fut. prof. jwl and m*a*p ade. psymaz of nuchwezi. *FigShare*, 2025. *Accessible via* https://doi.org/10.6084/M9.FIGSHARE.29064671.

[48] Joseph Willrich Lutalo. Platonic-exa-landscape.pdf — a basic example of full genome expression in the lu-genome expression system. https://figshare.com/articles/dataset/PLATONIC-EXA-landscape/_pdf_A_Basic_Example_of_Full_Genome_Expression_in_the_Lu-Genome_Expression_System/29876024/1, Aug 2025.

[49] Henry Cornelius Agrippa. *Three Books of Occult Philosophy*. Llewellyn Publications, Woodbury, Minnesota, 2014. Annotated edition (Edited by Donald Tyson) available at https://archive.org/details/three-books-of-occult-philosophy-henry-cornelius-agrippa-donald-tyson

[50] Carl G. Jung, Marie-Louise von Franz, Joseph L. Henderson, Aniela Jaffé, and Jolande Jacobi. *Man and His Symbols*. Aldus Books, London, 1964. Final editing by M.-L. von Franz after Jung's death.

[51] Joseph Willrich Lutalo. *Rock 'N' Draw*. I*POW, 2025. ***ISBN*** *978-9913-624-71-8 (First Edition). A physical copy might be accessed via the National Library of Uganda (NLU), but also, an education-purposes-only edition is accessible freely via Academia at* https://www.academia.edu/128601975/ROCK_N_DRAW_2025_.

[52] Joseph Lutalo. Concerning a transformative power in certain symbols, letters and words. *Academia*, 2025. The unstripped version accessible via: https://www.academia.edu/download/121530226/edtn_25FEB_Concerning_A_Transformative_Power_in_Certain_Symbols_Letters_and_Words.pdf.

[53] Raymond S. Nickerson. Counting, computing, and the representation of numbers. *Human Factors*, 30(2):181–199, 1988. Available at: https://www.researchgate.net/publication/258138626_Counting_Computing_and_the_Representation_of_Numbers.

[54] Joseph Willrich Lutalo. Numbers from arbitrary text: Mapping human readable text to numbers in base-36. *Academia.edu*, 2024. *Accessible via* https://www.academia.edu/123296302/Numbers_from_Arbitrary_Text_Mapping_Human_Readable_Text_to_Numbers_in_Base_36.

[55] Joseph Willrich Lutalo. Code: Computational mysticism ogf. https://figshare.com/articles/onlineresource/CODE_Computational_Mysticism_OGF/29858738/1, Aug 2025.

Applying TRANSFORMATICS In GENETICS

- ✓ 5 TABLES, 3 CORE PROBLEMS, 4 THEOREMS
- ✓ 12 DEFINITIONS, 3 LAWS
- ✓ 12 TRANSFORMERS, 18 TRANSFORMATIONS
- ✓ 4 ALGORITHMS, 2 CIPHER KEYS & MORE!
- ✓ COMPUTING GENETIC PROXIMITY VIA ADM
- ✓ GENETIC ANALYSIS WITH N-GRAMS, MSS & CHARACTERISTICS
- ✓ THE MATHEMATICS OF GENOME SEQUENCING
- ✓ THE LU-GENOME EXPRESSION SYSTEM & MORE!

JOSEPH WILLRICH LUTALO