Discrete Time 1-bit Music: foundations and models

by

Víctor Adán [a]

M.S., Massachusetts Institute of Technology (2005)

Submitted to the Music Department,
Graduate School of Arts and Sciences,
in partial fulfillment of the requirements for the degree of

Doctor of Musical Arts

at

Columbia University

May, 2010

[a]A citizen of MexiCorp, a wholly-owned subsidiary of USA Inc.

Discrete Time 1-bit Music: foundations and models

by Víctor Adán

Submitted to the Music Department,
Graduate School of Arts and Sciences, on May 2010,
in partial fulfillment of the requirements
for the degree of Doctor of Musical Arts

Abstract

This paper covers the theoretical bases of 1-bit music composition. Boolean algebra is introduced as the foundation of 1-bit music. Several models of 1-bit sequences are presented: deterministic models in the form of phasors and Xenakis sieves, and stochastic models in the form of finite state machines.
Music composition patterns (sequential and parallel composition, 1-bit counterpoint, modulation, hierarchies) are discussed within the context of 1-bit music.
Three common waveforms for encoding and transmitting 1-bit sequences (*line codes*) are reviewed and compared.
The paper concludes with a brief description of a series of pieces created with the models and tools presented.

Table of Contents

1 Introduction and Background **1**
- 1.1 Inspiration and Motivation . . . 1
- 1.2 Musical Representation . . . 2
  - 1.2.1 PCM (Pulse Code Modulation) . . . 2
- 1.3 What is Discrete-time 1-bit Music? . . . 3

2 1-bit Signals **5**
- 2.1 Pulse Code Modulation Waveforms . . . 5
- 2.2 The Subharmonic Series . . . 7

3 Symbolic 1-bit Sequences **9**
- 3.1 Boolean algebra . . . 10
  - 3.1.1 Boolean Vectors . . . 12

4 1-bit Sequence Models **15**
- 4.1 Boolean Impulses . . . 15
- 4.2 Boolean Cycles . . . 15
- 4.3 Boolean Phasors . . . 17
- 4.4 Xenakis Sieves . . . 18
  - 4.4.1 Analysis . . . 21
- 4.5 Non-periodicity and Randomness . . . 24
  - 4.5.1 Computational Mechanics . . . 28
  - 4.5.2 ϵ-machines . . . 30

5 1-bit Composition **33**
- 5.1 Parallel and Sequential Composition . . . 33
  - 5.1.1 Parallel (simultaneous) Perceptual Streams . . . 36
- 5.2 Modulation and Phasors . . . 38
- 5.3 Hierarchic stochastic finite state machines . . . 39
- 5.4 Fonoptera and the Tepozchiquilichtli Studies . . . 40

Appendix A Notation **49**

Appendix B 1-bit Instrument Construction **51**

Bibliography **57**

Acknowledgments

My thanks go to those who made this work, and all the work leading to it during my stay at Columbia University, possible.
Thank you to those professors with whom I studied and who supported me: Brad Garton, Fred Lerdahl, Fabien Levy, George Lewis, Tristan Murail, and Douglas Repetto.
Thank you to my CMC colleagues and collaborators Jeff Snyder, Daniel Iglesia and Sam Pluta for their inspiring work and feedback.
Special thanks go to Douglas Repetto for so generously sharing his time, knowledge and personal tools with me. My 1-bit instruments would not have happened without his help.
Last but not least, I thank Alejandra for her vast reserves of patience in dealing with the sometimes not so happy me.

CHAPTER ONE
Introduction and Background

1.1 Inspiration and Motivation

In 2002, Julio Estrada and I presented a paper and demonstration at the International Symposium on Musical Acoustics, held in Mexico City, on our implementation in *MúSIIC-Win 3.2 (Música: Sistema Interactivo de Investicación y Composición)* of Estrada's work on the combinatorial potential of scales and their application in the realm of time in the form of a wave shape generator.[6] In MúSIIC-Win 3.2, periodic wave shapes are constructed by partitioning a period into E equidistant segments and either assigning or not assigning an impulse to each segment. Essentially, all the compositions of E (in the mathematical sense)[1] are computed and rendered as a pulse wave for one to hear. The frequency of these periodic pulse waves can be changed in real time, so that one can audibly explore the relationship between the combinatorial arrangement of pulses within a cycle and the resulting timbre or rhythmic pattern.

Soon after, in 2004, while working at the MIT Media Lab, Noah Vawter aka "shifty" showed me his 1-bit microchip electronic instrument. I was not surprised by the sound, but I was struck by the clarity with which I could hear several concurrent streams. A couple of years later I decided to compose a 1-bit piece of music to see how far I could go with the most limited musical instrument conceivable.[2] How much music can one make with a 1-bit instrument?

[1]A composition of a positive integer n is tuple $\mathbf{a} = (a_1, a_2, \ldots, a_m)$ such that $a_1 + a_2 + \ldots + a_m = n$. i.e., it's an expression of n as a sum of strictly positive integers in which order matters. e.g., 5, $4+1$ and $1+4$ are all different compositions of 5.

[2]While exploring the use of dot matrix printers as 1-bit instruments and sharing my interest with other composers, I was made aware of the unique work of Tristan Perich, another composer doing a lot of 1-bit music. His music has since also been a source of inspiration.

1.2 Musical Representation

Humans have been encoding sound for thousands of years. The spoken word in the form of phonetic writing and the various systems of music notation are two broad examples. These systems were developed to represent a specific family of sounds (subsets of the universe of sounds) and to serve specific purposes.

Later came audio recording technology. By imprinting the atmospheric pressure changes on a variety of media, humans were now able to capture any arbitrary sound.[3] In Thomas Alva Edison's early phonographs, for example, the air pressure changes were mechanically recorded on tinfoil sheets wrapped around a grooved cylinder. Then came magnetic recordings; instead of making physical imprints, magnetic recorders converted the pressure changes into an analogous magnetic field along a moving tape. With the advent of digital technology, audio recording went back to the method of symbolic representation of the early days, with the notable difference that this time any sound could be encoded numerically.

1.2.1 PCM (Pulse Code Modulation)

The most common digital audio representation scheme used today is pulse-code modulations (PCM). PCM is a digital representation of quantized pulse-amplitude modulation (PAM).[14] In quantized PAM a signal is represented as a sequence of pulses of varying magnitudes that are equidistantly spaced over time (Figure 1-1). In sound recordings, these magnitude values are approximations of the atmospheric pressure changes registered by a microphone at regular time intervals. The accuracy of the approximation depends on the number of possible values available to represent each magnitude. This is typically given in bits (binary digits). In a 16-bit encoding (currently the standard for CDs) there are $2^{16} = 65536$ values available for each magnitude, in an 8-bit encoding there are $2^8 = 256$, in a 4-bit encoding there are $2^4 = 16$, with 2 bits there are $2^2 = 4$ values and with 1 bit there are $2^1 = 2$ values. The more bits used the more accurate the approximation of a continuous phenomenon such as air pressure changes. The smaller the encoding alphabet, the more inaccurate and distorted the representation will be.

[3]It is quite amazing that the multi-layered complexity that we hear in music can all be found embedded in the one dimensional signal which is the changing atmospheric pressure, and that it can all be captured by such a simple device as a microphone.

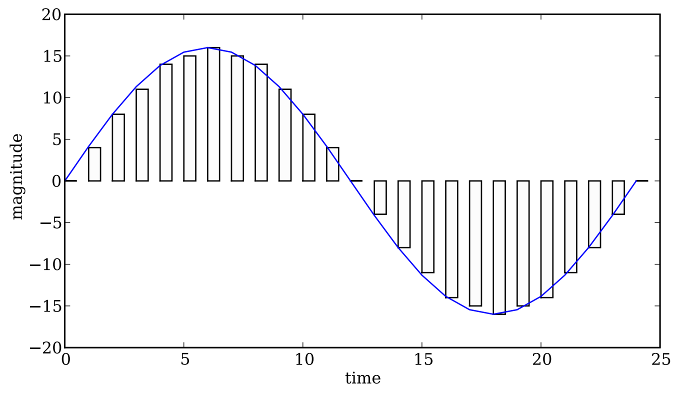


Figure 1-1: Quantized PAM representation of one cycle of a sine wave.

Once the magnitudes are measured, they are encoded numerically in binary form.

To reconstruct the PCM encoded sound, the reverse operation is performed. The symbolic binary words representing the evolving magnitudes are converted back into a physical reality by regenerating an electrical signal that varies in relation to these magnitudes. This discrete signal is then smoothed with a high-pass filter to obtain a continuously varying signal.

1.3 What is Discrete-time 1-bit Music?

In 1-bit music there are only two "words" in the musical alphabet. i.e., at any given instant there is one and only one of two events: '0' or '1'. Thus, the set of all possible events available at any given instant can be represented with just one bit.

In addition, two *time-points* or events can not be arbitrarily close to each other. i.e., there is a minimum time interval $T = \Delta t$ between events. This smallest time interval is called the *tatum* (time atom) and all time intervals are integer multiples of it.

One of the most interesting things about 1-bit music composition is that one is directly confronted with the problem of creating pitches, timbres and polyphony (parallel perceptual streams), all from a single train of pulses. With only two symbols (0 and 1) there is no "vertical" information, no subtlety in the degree of push and pull. Thus, in the confines of a 1-bit music, one is forced to use time to convey all information; time is the only carrier of information. i.e., everything must be created from *rhythm*.[4] Rhythm is the *sine qua non* of music.

[4]In music, and from a psychoacoustic perspective, rhythm is usually contrasted with pitch, the first referring to our perception of events that happen at a rate of less than 20 Hz, approximately, and the later to those events that happen at a rate greater than 20 Hz. Rhythm is here understood in a more general sense as the distribution of events over time, regardless of speed.

CHAPTER TWO
1-bit Signals

2.1 Pulse Code Modulation Waveforms

As introduced in the previous chapter, PCM is a way of encoding sound into a binary representation. In order to hear the recording, the binary encoding is converted back into a smooth continuous signal to recover the originally continuous air pressure changes. 1-bit music is, however, by definition, a discrete two state universe. How then, should a 1-bit sequence be converted into a physical reality? Certainly, the physical signal generated from a binary sequence must also be subject to the bounds of the 1-bit world. Strictly, a 1-bit sequence (10101010 · · ·) would cease to be 1-bit if it were interpolated and transmitted as a continuous signal (Figure 2-1). A 1-bit waveform describing electrical or mechani-

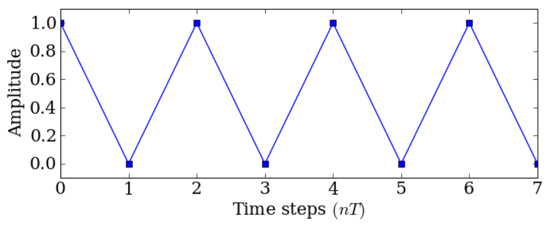


Figure 2-1: A 1-bit sequence (10101010 · · ·) rendered as a continuous waveform.

cal variations should, ideally, have but two states, just like the abstract 1-bit sequences. Idealized pulse waves with just two amplitudes (a maximum and a minimum) satisfy this requirement. A pulse wave used to carry binary digits (e.g., through electrical wire) is called a pulse-code modulation waveform (PCM waveform). A variety of PCM waveforms (also called *line codes*) exist. Figure 2-2 shows three commonly used line codes: NRZ-L, NRZ-M and Unipolar RZ.[14]

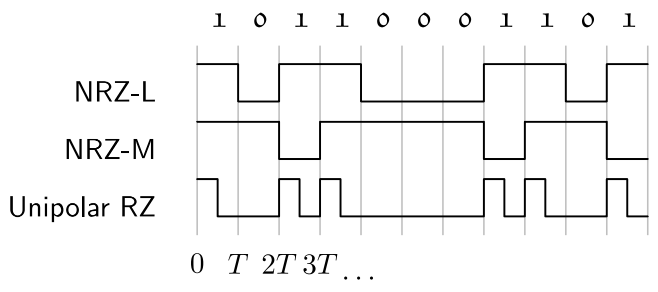
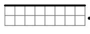
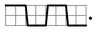
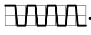


Figure 2-2: Three PCM waveforms commonly used to transmit binary data.

In NRZ-L (Nonreturn-to-zero Level), 1 is represented as a high voltage level and a 0 is represented as a low voltage level. In NRZ-M (Nonreturn-to-zero Mark), 1 (the mark) is represented as a change in voltage level, 0 is represented by no change in level. In Unipolar RZ (Return-to-zero), 1 is represented as a short high voltage pulse, 0 is represented by the absence of a pulse. In Unipolar RZ, the duration of the pulse carries no information and varies depending on the application (naturally it is always a fraction of T). Its purpose can be simply to make the pulse detectable by a receiver or to provide enough power to drive an electrical device.

In NRZ-M and Unipolar RZ, 1 is encoded as an onset. In NRZ-M, a change in voltage occurs every time there is a 1 and only when there is a 1. In Unipolar RZ a pulse is triggered every time there is a 1 and only in the presence of a 1. In NRZ-L, a change of voltage occurs only when there is a change of value. Thus, in NRZ-L a sequence (1111 $\cdots$) will yield a constant high value  $\cdots$, while NRZ-M yields  $\cdots$ and Unipolar RZ yields  $\cdots$.

All three codes will sound differently when sent to a speaker. If 1 is assumed to be the onset of a perceivable event (like a percussion attack) then line codes such as NRZ-M or Unipolar RZ are used.

In 1-bit music, the PCM waveform is not just a digital transmission scheme, it is, in a sense, *the* music; it is what we actually hear when the wave —in the form of voltage variations— is sent directly to a speaker to be converted into sound pressure changes.

2.2 The Subharmonic Series

Not every frequency can be perfectly reproduced in the world of discrete time 1-bit music. Since the tatum is the smallest possible time interval between two consecutive impulses, the highest frequency that can be reproduced in such a system is $\frac{1}{T}$ for a Unipolar RZ PCM waveform and $\frac{1}{2T}$ for NRZ. Because all time intervals must be multiples of the tatum, the next highest frequency possible is $\frac{1}{2T}$ for Unipolar RZ and $\frac{1}{4T}$ for NRZ. All other possible frequencies follow logically: $\frac{1}{nT}$, for $n \in \mathbb{N}^*$. Thus, the frequencies that can be reproduced within the confines of discrete time 1-bit music are those of the subharmonic series: $\{1/1, 1/2, 1/3, 1/4, \cdots\}$. The higher the frequency range, the smaller the set of available frequencies. Between $\frac{1}{T}$ and $\frac{1}{2T}$ (the range of an octave, in pitch parlance) inclusive, there are two available frequencies: $\frac{1}{T}$ and $\frac{1}{2T}$. Between $\frac{1}{2T}$ and $\frac{1}{4T}$ (again a ratio of 1:2) there are three. Between $\frac{1}{4T}$ and $\frac{1}{8T}$ there are five, and so on. In general, between $\frac{1}{n}$ and $\frac{1}{2n}$, inclusive, there are $n+1$ available frequencies:

$$\overbrace{(1,2)}^{2^0+1},\ \overbrace{(2,3,4)}^{2^1+1},\ \overbrace{(4,5,6,7,8)}^{2^2+1},\ \overbrace{(8,9,10,11,12,13,14,15,16)}^{2^3+1},\ \cdots$$

Further, since the timbre is determined by the pulse pattern of the cycle (in the case of periodic sounds with a fixed frequency > 20 Hz), and the shorter the period, the smaller the set of available cycle patterns (see Section 4.2), the higher the frequency the smaller the set of available timbres.

In what remains of this paper I will discuss only the symbolic dimension of 1-bit music.

CHAPTER THREE
Symbolic 1-bit Sequences

Mixing digital audio involves simple numeric addition. When two 16-bit signals (byte vectors) $\overline{a}$ and $\overline{b}$ are mixed, their values corresponding to the same time index n are added ($\overline{a}[n] + \overline{b}[n]$), resulting in a new value $\overline{c}[n] = \overline{a}[n] + \overline{b}[n]$.

Example. If $\overline{a} = (1, 2, 3, 4)$ and $\overline{b} = (1, 0, 1, 0)$, then

$$\overline{a} + \overline{b} = (1, 2, 3, 4) + (1, 0, 1, 0) = (2, 2, 4, 4).$$

The finite alphabet of 2^{16} words limits the result of the sum to $65536 - 1$. If the sum of two or more values exceeds this limit, the result is "clipped" and set to $2^{16} - 1$.[1]

How does mixing work in 1-bit music? $1 + 1 = 2$ is not an option because the alphabet of 1-bit music comprises only the two words $\mathtt{1}$ and $\mathtt{0}$, so $\mathtt{1} + \mathtt{1}$ must result in either $\mathtt{1}$ or $\mathtt{0}$. Assuming $\mathtt{1}$ represents a perceivable event and $\mathtt{0}$ the absence of a perceivable event, we might want $\mathtt{1} + \mathtt{1} = \mathtt{1}$. What should $x + y$ yield for the other three possible arrangements of $\mathtt{1}$ and $\mathtt{0}$? To maintain consistency with arithmetic addition, we might define our 1-bit + as follows:

$$\begin{aligned} \mathtt{0} + \mathtt{0} &= \mathtt{0} \\ \mathtt{0} + \mathtt{1} &= \mathtt{1} \\ \mathtt{1} + \mathtt{0} &= \mathtt{1} \\ \mathtt{1} + \mathtt{1} &= \mathtt{1} \end{aligned}$$

[1]This is the famous "clipping" digital distortion.

The definition of a binary operator (in this case $+$) as an exhaustive listing of all possible assignments of 1 and 0 to operands x and y, together with their corresponding outputs is called the *truth table* of the operator.

Clearly, this is one of many ways in which the operator could have been defined. This exercise begs the question of what other binary operators might be defined that could be useful in 1-bit music composition. Since we are dealing with a set of only two values, 1 and 0, we can actually write out and enumerate all 2^4 possible "interactions" between two variables x and y, with $x, y \in \{1, 0\}$. Table 3.1 shows all 16 possible outputs for the four possible arrangements of the two letters 1 and 0. Note that

x	y	*Truth Tables*															
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
		1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*

Table 3.1: All 2^4 possible ways of defining binary operations between the two values 1 and 0.

the 8th column corresponds to our definition of $+$. Fifteen other binary operations are laid out in the table. Could any of those also be useful in a 1-bit music toolbox? Are the sixteen truth tables related in any "logical" way?

These questions point to the more general problem of defining some form of two-valued algebraic system. Naturally, this work has already been done by several people, most notably George Boole in his 1854 book *An Investigation of the Laws of Thought*[2]. In the book, Boole sets the foundations for what is now called Boolean Algebra: the algebra of thought, logic, classes, digital circuits and... 1-bit music.

3.1 Boolean algebra

In general, an algebra is composed of a set $\mathcal{A}$ and a collection of operators that act on $\mathcal{A}$.[8] Boolean algebra is the algebra of two values. In 1-bit music only two values are needed, so I will stick to the prototypical boolean set $\mathcal{A} = \{1, 0\}$. The operators acting on this set can be defined axiomatically from the truth tables in Table 3.1. Because we still don't have a name or symbol for each of the sixteen definable binary operators, I will refer to them by number as $\mathrm{op}(i)$, where i is the number of the corresponding truth table found in Table 3.1. The provisional definition of $+$ given above would then be written as

$$\begin{aligned} 0 \,\mathrm{op}(8)\, 0 &= 0 \\ 0 \,\mathrm{op}(8)\, 1 &= 1 \\ 1 \,\mathrm{op}(8)\, 0 &= 1 \\ 1 \,\mathrm{op}(8)\, 1 &= 1 \end{aligned}$$

Note that some of the truth tables are of no use as binary operators. The first (definition of op(1)) and last (definition of op(16)) are simply 0 and 1 respectively; no matter what the values for x and y are, these two will always return the same constant. Truth table 4 is always x and truth table 6 is always y. i.e., $x \,\mathrm{op}(4)\, y = x$ and $x \,\mathrm{op}(6)\, y = y$. Notice that the truth tables are inversely mirrored at the center of their arrangement in Table 3.1, so that the last truth table is the inverse of the first, the next-to-last is the inverse of the second, and so on. From this observation we can define a handy unary operator '$\neg$' called the "complement" or "negation". Thus, $\neg 1 = 0$ and $\neg 0 = 1$. In general, $x \,\mathrm{op}(i)\, y = \neg(x \,\mathrm{op}(16+1-i)\, y)$ for $1 \leq i \leq 16$. This means that half of the operators can be defined in terms of the other via negation. This also implies that if $\mathrm{op}(i)$ is a dispensable binary operator (in the sense that only one or none of the two inputs x and y defines the output) so is $\mathrm{op}(16+1-i)$. Removing the four dispensable truth tables we are left with ten useful binary operators, but since one half can be defined in terms of the other via negation, we are left with, what at this point appears to be, five axiomatic binary operators. Table 3.2 again shows all the truth tables for all possible binary operators, this time with the standard symbols used for the six most commonly defined operators.

Within the six standard symbols that appear in Table 3.2 there exists only one mirroring correspondence via $\neg$: that between $\oplus$ and $\equiv$. Thus, one can be defined in terms of the other: $x \equiv y = \neg(x \oplus y)$. This leaves us with five axiomatic binary operators and one unary operator. But the set can further be reduced. The standard axiomatic definition of the boolean operators takes $\wedge$, $\vee$ and $\neg$ (AND, OR and NOT, respectively)

x	y	*Truth Tables*															
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
		1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*
			$\wedge$					$\oplus$	$\vee$		$\equiv$		$\Leftarrow$		$\Rightarrow$		

Table 3.2: Of the 2^4 boolean binary operations available, four can be discarded because their output is defined either only by one or none of the values of the variables x and y. These are marked in gray. The symbols under truth tables 2, 7, 8, 10, 12, 14 are the standard symbols associated with each of those operators.

as the three basic operators from which all others are derived:[2]

$$
\begin{aligned}
x \oplus y &= (x \vee y) \wedge \neg(x \wedge y) \\
x \Rightarrow y &= \neg(x \wedge \neg y) \\
x \Leftarrow y &= x \vee \neg y
\end{aligned}
$$

3.1.1 Boolean Vectors

Definition 1 (Boolean vector)**.** A boolean vector $\overline{x} = (x_1, x_2, \ldots, x_m)$ is a finite length tuple with $x_i \in \{1, 0\}$, and $m \in \mathbb{N}^* < \infty$.

The boolean algebra with set $\mathcal{A} = \{0, 1\}$ is called the prototypical boolean algebra. More generally, however, a boolean algebra can be applied to a variety of other sets with more than two elements. The step from the set $\mathcal{A} = \{0, 1\}$ of two boolean values to the set $\mathcal{A}^3 = \{(000), (001), (010), (011), (100), (101), (110), (111)\}$, for example, is trivial. For the unary operator $\neg$, $\neg\overline{x}$ simply applies the negation to each element of $\overline{x}$.

Example. If $\overline{x} = (101)$, then $\neg\overline{x} = (010)$.

For binary operators, the operation is applied to bit pairs with corresponding indexes. i.e., If $\overline{x} = (x_1, x_2, \ldots, x_m)$ and $\overline{y} = (y_1, y_2, \ldots, y_m)$,

[2]As an exercise, the reader can verify that these equalities are true.

then $\overline{x} \wedge \overline{y} = (x_1 \wedge y_1, x_2 \wedge y_2, \ldots x_m \wedge y_m)$. In this case, both vectors $\overline{x}$ and $\overline{y}$ must have the same length.

Example. If $\overline{x} = (101)$, and $\overline{y} = (110)$, then $\overline{x} \wedge \overline{y} = (100)$.

Definition 2 (Concatenation operator: $|$)**.** For any two finite length vectors $\overline{x} = (x_1, x_2, \ldots, x_m)$ and $\overline{y} = (y_1, y_2, \ldots, y_n)$.

$$\overline{x} \mid \overline{y} = (x_1, x_2 \ldots, x_m, y_1, y_2, \ldots, y_n)$$

i.e., the concatenation operator combines two vectors of length n and m into one vector of length $n + m$.

CHAPTER FOUR 1-bit Sequence Models

4.1 Boolean Impulses

Definition 3 (Symbolic Impulse)**.** A symbolic impulse is here defined simply as a boolean True: 1.

Definition 4 (Null)**.** A *null* is a boolean False: 0.

Definition 5 (Impulse train)**.** An impulse train is a boolean vector. i.e., an ordered collection of impulses and/or nulls.

Definition 6 (Inter-impulse-interval)**.** Let $\overline{x} = (x_1, x_2, \ldots, x_m)$ be an impulse train with impulses at indexes i and k. An inter-impulse-interval exists between impulses x_i and x_k iff there does not exist a j, with $i < j < k$, such that x_j is an impulse. The inter-impulse-interval between impulses x_i and x_k is defined as the absolute value of the difference between indexes i and k: $\mid i - k \mid$.

Definition 7 (Delta-impulse function)**.** Given an impulse train $\overline{x} = (x_1, x_2, \ldots, x_m)$, a delta-impulse function $\Delta(\overline{x})$ returns a tuple of all the inter-impulse-intervals in $\overline{x}$.

Example. Let $\overline{x} = (10010)$, then $\Delta(\overline{x}) = (3, 2)_\Delta$. The subscript Δ is used to emphasize the fact that these are indeed inter-impulse-intervals.

4.2 Boolean Cycles

There are 2^m possible arrangements of 1s and 0s for a boolean sequence of length m. For $m = 3$, for example, the following 8 sequences are possible: (000), (001), (010), (011), (100), (101), (110), (111). Some of these are, however, not cyclically distinct. Taking each as a closed loop, like a necklace, we find the following equivalences:

$$(001) \sim_{\circlearrowleft} (010) \sim_{\circlearrowleft} (100)$$
$$(011) \sim_{\circlearrowleft} (101) \sim_{\circlearrowleft} (110)$$

More generally, using cycle notation, we have that: $(a)(b)(c) = (abc) = (acb)$. i.e., the identity permutation —the permutation that leaves all elements in their original place— $(a)(b)(c)$ is equivalent to the permutation (abc) —which maps a to b, b to c, and c to a— and the permutation (acb) —which maps a to c, c to b, and b to a. (000) and (111) are unique. Thus, there are four cyclically distinct boolean vectors of length three. Removing the null sequence (000) from the set leaves us with only three useful boolean cycles of length $m = 3$ (see Figure 4-1).

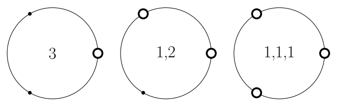


Figure 4-1: The only three distinct boolean cycles of length 3 with at least one impulse.

Definition 8 (Boolean Cycle)**.** A boolean cycle $\overline{x}^p_{\circlearrowleft} = (x_1, x_2, \ldots, x_m)^p_{\circlearrowleft}$ is a boolean sequence of finite length m that has no beginning and no end. It is a closed loop structure that can be indexed and traversed infinitely in one direction or the other. The reference phase p sets the 0^{th} index position to the $(p+1)^{th}$ element of the cycle. Thus, $0 \leq p < \text{len}(x)$ always. When absent, p is assumed to be 0.

Example. The cycle $\overline{x}^0_{\circlearrowleft} = (100)^0_{\circlearrowleft} = (\cdots 100100 \cdots)$ has 1 at phase 0, 0 at phase 1 and 0 at phase 2. Using index notation, $\overline{x}^0_{\circlearrowleft}[0] = 1$, $\overline{x}^0_{\circlearrowleft}[1] = 0$ and $\overline{x}^0_{\circlearrowleft}[2] = 0$. Indexes can go to infinity: $\overline{x}^0_{\circlearrowleft}[3] = 1$, $\overline{x}^0_{\circlearrowleft}[4] = 0$, ... The element returned is the $((n+p) \mod \text{len}(\overline{x}))^{th}$ element of the cycle. For $\overline{x}^1_{\circlearrowleft} = (100)^1_{\circlearrowleft} = (\cdots 001001 \cdots)$, $\overline{x}^1_{\circlearrowleft}[0] = 0$, $\overline{x}^1_{\circlearrowleft}[1] = 0$, $\overline{x}^1_{\circlearrowleft}[2] = 1$, $\overline{x}^1_{\circlearrowleft}[3] = 0$ and so on. Since $\overline{x}^0_{\circlearrowleft}$ and $\overline{x}^1_{\circlearrowleft}$ are cyclically symmetric, we can write $\overline{x}^0_{\circlearrowleft} \sim_{\circlearrowleft} \overline{x}^1_{\circlearrowleft}$.

Definition 9 (Irreducible cycle)**.** A boolean cycle $\overline{x}^p_{\circlearrowleft} = (x_1, x_2, \ldots, x_m)^p_{\circlearrowleft}$ is an irreducible cycle iff for all $q \in \{1, 2, \cdots, \text{len}(\overline{x}^p_{\circlearrowleft}) - 1\}$.

$$\overline{x}^0_{\circlearrowleft} \equiv \overline{x}^q_{\circlearrowleft} \neq (\cdots 111 \cdots)$$

Example. Let $\overline{x}^0_{\circlearrowleft} = (110110)^0_{\circlearrowleft}$. $\overline{x}_{\circlearrowleft}$ is not an irreducible cycle because $\overline{x}^0_{\circlearrowleft} \equiv \overline{x}^3_{\circlearrowleft} = (111111)$. The cycle can be reduced to the length-three cycle $\overline{y}^0_{\circlearrowright} = (110)^0_{\circlearrowright}$. $\overline{y}_{\circlearrowright}$ is irreducible because $\overline{y}^0_{\circlearrowright} \equiv \overline{y}^1_{\circlearrowright} = (100)$ and $\overline{y}^0_{\circlearrowright} \equiv \overline{y}^2_{\circlearrowright} = (010)$, none of which are (111).

4.3 Boolean Phasors

In composition it is generally useful and thus desirable to orthogonalize parameters. In the domain of sound we separate pitch (the main perceptual correlate of frequency) from timbre (highly determined by the waveshape and relatively independent of pitch). Here we want to separate the length of a cycle from the cyclical pattern itself. i.e., we want to factor out the length of a cycle in order to draw equivalences across cycles of different lengths.

Take the sets of boolean cycles with lengths 4 and 6, for example (see Figures 4-2 and 4-3). Cycle $(1000)_{\circlearrowright}$ and $(100000)_{\circlearrowright}$ have different lengths but the same inter-impulse-interval pattern: a single impulse. Thus, we would want these to be equivalent: $(1000)_{\circlearrowright} \sim_{\circlearrowright} (100000)_{\circlearrowright}$. In delta notation $(4)_{\Delta\circlearrowright} \sim_{\circlearrowright} (6)_{\Delta\circlearrowright}$. The cycles of $(1010)_{\circlearrowright}$ and $(100100)_{\circlearrowright}$ are both symmetrically divided by two impulses in two equal halves. Thus, $(1010)_{\circlearrowright} \sim_{\circlearrowright} (100100)_{\circlearrowright}$, or equivalently $(2,2)_{\Delta\circlearrowright} \sim_{\circlearrowright} (3,3)_{\Delta\circlearrowright}$. The equivalence $(1111)_{\circlearrowright} \sim_{\circlearrowright} (111111)_{\circlearrowright}$ seems to be of a slightly different nature. $(111111)_{\circlearrowright}$ has more impulses than $(1111)_{\circlearrowright}$, but since the inter-impulse-intervals are identical, there is no way to know from which of the two cycles the sequence $(\cdots 1111 \cdots)$ might have come from. Thus, $(1)_{\circlearrowright} \sim_{\circlearrowright} (11)_{\circlearrowright} \sim_{\circlearrowright} (111\cdots)_{\circlearrowright}$. Actually, all the cyclical patterns so far discussed are equivalent since they can all be reduced to the cycle $(1)_{\circlearrowright}$ by inter-impulse-interval scaling or *reduction* (i.e., by shortening a non-irreducible cycle to its irreducible form.) Thus, $(1)_{\circlearrowright} \sim_{\circlearrowright} (11)_{\circlearrowright} \sim_{\circlearrowright} (111\cdots)_{\circlearrowright} \sim_{\circlearrowright} (1010)_{\circlearrowright} \sim_{\circlearrowright} (101010\cdots)_{\circlearrowright} \sim_{\circlearrowright} (100100)_{\circlearrowright} \sim_{\circlearrowright} (100100100\cdots)_{\circlearrowright} \sim_{\circlearrowright} \ldots$

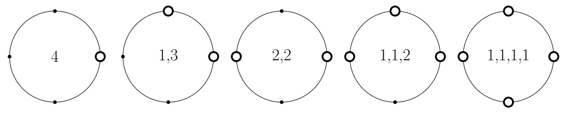


Figure 4-2: All distinct boolean cycles of length 4, excluding the null cycle.

Clearly, all boolean cycles of length l will find an equivalence in the set of boolean cycles of length nl, for $n \in \mathbb{N}^*$.

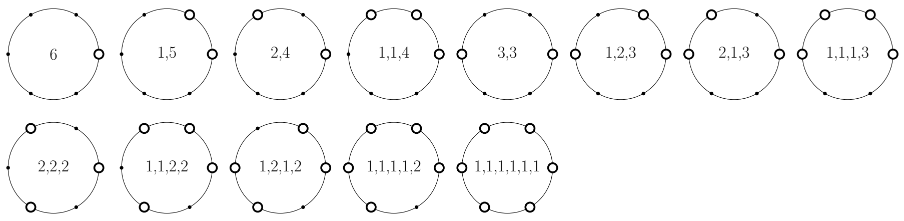


Figure 4-3: All distinct boolean cycles of length 6, excluding the null cycle.

Definition 10 (Boolean Phasor)**.** A boolean cycle $\overline{x}_{\circlearrowleft} = (x_1, x_2, \ldots, x_m)_{\circlearrowleft}$ is a boolean phasor[1] $\overline{x}_{\phi}$ iff it satisfies the following properties:

1. $\overline{x}_{\circlearrowleft}$ is an irreducible cycle.

2. The inter-impulse-intervals of the cycle do not share a common divisor other than one: $\mathrm{GCD}(\Delta(\overline{x}_{\circlearrowleft})) = 1$. Equivalently, $\nexists\ \overline{y}_{\circlearrowleft}$ such that $n\Delta(\overline{y}_{\circlearrowleft}) = \Delta(\overline{x}_{\circlearrowleft})$, for any $n \geq 2$.

Figure 4-4 shows all boolean phasors with cycle lengths 1, 3, 4, 5, 6 and 7. All boolean cycles of length 2 are redundant (included in the single length 1 phasor) and thus absent. Note that the five boolean cycles of length 4 reduce to two boolean phasors, and the thirteen boolean cycles of length 6 reduce to seven phasors.

4.4 Xenakis Sieves

> ... every well-ordered set can be represented as points on a line, if it is given a reference point for the origin and a length u for the unit distance, and this is a sieve.[17]

There are a couple of equivalent ways in which sieves can be formalized. Xenakis' universe and point of departure is the set of integer numbers $\mathbb{Z}$. Using residue classes and combining them via set operators, Xenakis constructs asymmetric subsets of $\mathbb{Z}$.

[1]The name "phasor" is naturally suggested by the use of the word to refer to a rotating vector represented as a complex exponential.

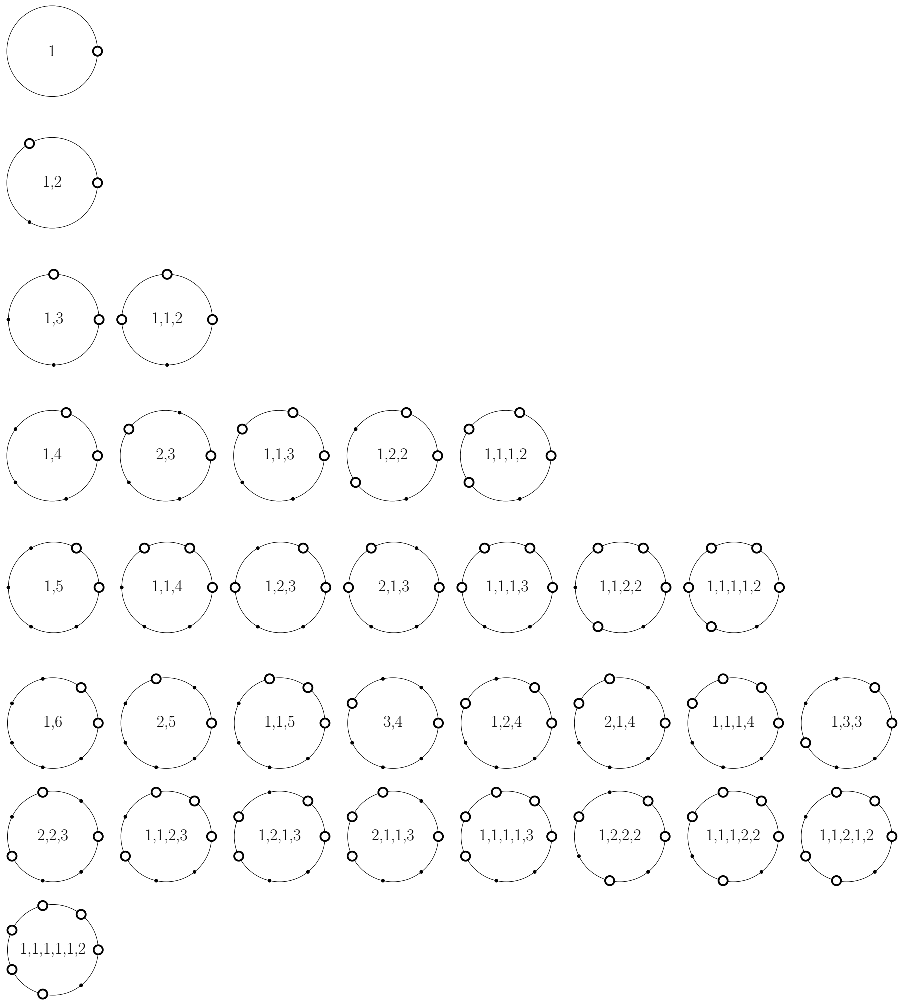


Figure 4-4: All boolean phasors with cycle lengths 1 through 7.

In [16], Xenakis defines an elementary sieve as a *residue class*, an equivalence relation of congruence modulo m.[2]

Definition 11 (Congruence modulo m)**.** Two integers c and r are said to be congruent modulo m if there exists an integer q such that $c = qm + r$. This is written $c \equiv r \pmod{m}$.

Definition 12 (Residue class)**.** The set of all integers c congruent r modulo m is called a *residue class* modulo m, here denoted $\mathrm{rc}(m, r)$.

$$\mathrm{rc}(m, r) = \{x \in \mathbb{Z} : x = qm + r\}$$

Example. The residue classes $\mathrm{rc}(12, i)$ for $i \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$ correspond to the pitch class sets Western indoctrinated musicians know and love. Pitch class 0 (C): $\mathrm{rc}(12, 0) = \{\cdots, -12, 0, 12, 24, 36, \cdots\}$; pitch class 1 (C♯): $\mathrm{rc}(12, 1) = \{\cdots, -11, 1, 13, 25, 37, \cdots\}$; etc.

Definition 13 (Complex Sieve)**.** A *complex sieve* is a subset of $\mathbb{Z}$ that can not be expressed as a single residue class and that must be expressed as a combination of multiple elementary sieves via set operators like union ($\cup$), intersection ($\cap$), complementation and exclusive disjunction ($\oplus$).[3]

Example (union)*.* The union of $\mathrm{rc}(2, 0)$ and $\mathrm{rc}(3, 0)$:

$$\begin{aligned} \mathrm{rc}(2,0) \cup \mathrm{rc}(3,0) &= (\cdots, -2, 0, 2, 4, 6, 8, \cdots) \cup (\cdots - 3, 0, 3, 6, 9, \cdots) \\ &= (\cdots, -3, -2, 0, 2, 3, 4, 6, 8, 9, \cdots) \end{aligned}$$

Example (exclusive disjunction)*.*

$$\mathrm{rc}(2,0) \oplus \mathrm{rc}(3,0) = (\cdots, -2, 2, 3, 4, 8, 9, 10, \cdots)$$

As abstract integer subsets, sieves can be applied to a variety of domains. In particular, these integer subsets can be thought of as indexes to a boolean vector, where all the values corresponding to these indexes are 1, and 0 otherwise.

[2]In his various texts, Xenakis uses different kinds of notation to represent a residue class. In [16], he notates m_r, the modulo subscripted with the residue, while in [17] he uses the notation (m, r). Here I will use the notation $\mathrm{rc}(m, r)$ to make the residue class more explicit.

[3]Xenakis appears to have never used the XOR (exclusive disjunction) operator.

Example. Considering only the positive integers, $\text{rc}(2,0) \cup \text{rc}(3,0) = (0,2,3,4,6,8,9,\cdots)$. Converting this ordered set into a boolean vector $\overline{x}$, where $\overline{x}[i] = 1$ for $i \in \{0,2,3,4,6,8,9,\cdots\}$, 0 otherwise, yields $(101110101110\cdots)$.

This perspective moves us away from thinking about sieves in terms of integer subsets and brings us closer to the world of boolean vectors previously discussed.

4.4.1 Analysis

A parallel can be drawn between Fourier analysis and boolean representations of Xenakis sieves. Here too, a complex sequence can be decomposed into a collection of simple periodic functions, each with a different period and/or phase. Inversely, and as we have already seen, a complex sequence can be constructed from a combination of simple bases.

Definition 14 (Sieve basis function)**.** A sieve basis function $\delta : \mathbb{Z} \rightarrow \{0, 1\}$ maps a residue class $\text{rc}(m, r)$ to a periodic boolean vector:

$$\delta_m^r[n] \triangleq \begin{cases} 1, & \text{if } n \equiv r \pmod{m} \\ 0, & \text{otherwise.} \end{cases}$$

The notation δ_m^r will be used when indexing is not required.

To emphasize the distinction between sets of residue classes and these boolean sequences, I shall use the standard boolean operators when dealing with δ_m^r functions. The above examples would then be expressed as:

$$\delta_2^0 \vee \delta_3^0 = (\cdots 101110 \cdots)$$

$$\delta_2^0 \oplus \delta_3^0 = (\cdots 001110 \cdots)$$

In his 1990 *Sieves* article[17], Xenakis discusses both the construction of complex sequences through the combination of elementary sieves and the inverse transformation: the decomposition of a complex sequence into elementary sieves. The algorithm he presents might be called a XOR decomposition because the δ_m^r functions obtained from the decomposition are orthogonal (i.e., mutually exclusive since the intersection of their corresponding residue classes yields the empty set) and thus can be XORed to obtain the original complex sequence. For example, the sequence $(101110\cdots)$ would be decomposed into the two bases δ_2^0 and

δ_6^3: $(101010\cdots) \oplus (000100\cdots)$. Jones[9] considers this XOR decomposition to be "flawed" mainly because it excludes overlapping in favor of orthogonality.[4] This is not a bad thing *per se* of course, but one might indeed want the decomposition of $(101110\cdots)$ to yield all the periodicities found in the sequence, in this case δ_2^0 and δ_3^0. This can be called an OR decomposition because the δ_m^r basis obtained can be ORed together to recover the original complex sieve.

Just as the Fourier transform can decompose a signal into sinusoidal functions by measuring the correlation between the sinusoidal basis and the signal, so too the OR and XOR boolean decompositions express a complex boolean sieve as a combination of elementary boolean basis functions δ_m^r.

The correlation between a boolean vector $\overline{s}$ and a sieve basis function δ_m^r is given by

$$\langle \overline{s}, \delta_m^r \rangle = \frac{1}{\sum_{n=0}^{\mathrm{len}(\overline{s})-1} \delta_m^r[n]} \sum_{n=0}^{\mathrm{len}(\overline{s})-1} \overline{s}[n]\delta_m^r[n] \tag{4.1}$$

Note that the boolean 1 values are here treated as numeric 1s to perform the sum, and 0 as numeric zeros. Thus the correlation function maps booleans to rationals. $\langle \overline{s}, \delta_m^r \rangle$ tells us how much δ_m^r correlates with $\overline{s}$. i.e., how much of δ_m^r is in $\overline{s}$. Notice that $\langle \overline{s}, \delta_m^r \rangle$ is normalized, so $0 \leq \langle \overline{s}, \delta_m^r \rangle \leq 1$ always. When $\langle \overline{s}, \delta_m^r \rangle = 1$, the periodic delta function δ_m^r correlates perfectly with sequence $\overline{s}$. When $\langle \overline{s}, \delta_m^r \rangle = 0$, $\overline{s}$ and δ_m^r are orthogonal, meaning that there is not a single index n for which both function have a value 1.

All sieves, elementary or complex, are periodic. Thus $\overline{s}$ has a period N, which means that $\mathrm{len}(\overline{s})$ can be made to be mN. Then, the correlation

[4]In [9], Jones proposes an alternative algorithm for sieve decomposition that offers a compromise between exactness and compact representation. It is a lossy OR decomposition that reconstructs an approximation to a complex sieve by combining the extracted basis via the OR operator.

function can be written as

$$\begin{aligned}\langle \overline{s}, \delta_m^r \rangle &= \frac{m}{mN} \sum_{n=0}^{mN-1} \overline{s}[n]\delta_m^r[n] \\ &= \frac{1}{N} \sum_{n=0}^{mN-1} \overline{s}[n]\delta_m^r[n]\end{aligned}$$

Further, since δ_m^r is zero except when $n \equiv r \pmod{m}$ then the equation becomes

$$\langle \overline{s}, \delta_m^r \rangle = \frac{1}{N} \sum_{n=0}^{N-1} \overline{s}[r + nm].$$

This boolean correlation analysis is akin to the *periodicity transform* discussed by Sethares in [11]. In contrast to the Fourier transform, which is a function of frequency, this transformation is a function of period m and phase r.

Definition 15 (Fuzzy Sieve Spectrum)**.** The spectrum of a boolean vector $\overline{s}$ with period N is the set of all correlation values $\langle s, \delta_m^r \rangle$ for every period m and phase r, such that $0 \le m \le N$ and $0 \le r < m$, and m is divisor of N.

Example. Compute the fuzzy sieve spectrum of $\overline{s} = \delta_2^0 \vee \delta_3^0 = (101110)$.

m							
1	4/6	0	0	0	0	0	
2	1	1/3	0	0	0	0	
3	1	1/2	1/2	0	0	0	
6	1	0	1	1	1	0	
	0	*1*	*2*	*3*	*4*	*5*	r

Notice in the table that periods 4 and 5 are skipped because 4 and 5 are not divisors of 6. Since 2 and 3 are divisors of 6, and for these moduli there exists a phase $r = 0$ such that $S[2, 0] = S[3, 0] = 1$, then indeed $\overline{s}$ is decomposable into $\delta_2^0 \vee \delta_3^0$.

The fuzzy sieve spectrum tells us how much each basis function correlates with a given sequence. If we are only interested in perfect correlation (the 1s in the table above) then the correlation function can be made strictly boolean:

Definition 16 ((Boolean) Sieve Spectrum)**.** For all m divisor of N,

$$\langle s, \delta_m^r \rangle_{\mathbb{B}} = \bigwedge_{n=0}^{N/m-1} \overline{s}[r + nm]$$

Example. The sieve spectrum of $\overline{s} = \delta_2^0 \vee \delta_3^0 = (101110)$:

m \ r	*0*	*1*	*2*	*3*	*4*	*5*
1	0	0	0	0	0	0
2	1	0	0	0	0	0
3	1	0	0	0	0	0
6	1	0	1	1	1	0

4.5 Non-periodicity and Randomness

Periodic sequences of any length can be compactly represented with sieves. Noise, however, is not periodic. In order to construct a boolean vector that appears to be noise, several sieve bases must be combined. Further, at most one cycle of the resulting expression can be used. Just as the Fourier transform fails to provide a compact representation of noise—in the sense that all coefficients for all frequencies within a given bandwidth will be greater than zero—so too a sieve decomposition of a finite noisy sequence will result in an expression containing many non-zero bases. As the length of the sequence tends to infinity, so will the number of sieve bases necessary to reconstruct the sequence. Thus, sieves are not the best tool for modeling boolean noise—at least in terms of compact representation. Other models are better suited for non-periodic/non-deterministic sequences.

To construct a random sequence of 1s and 0s one can simply toss a coin as many times as necessary and record the output of each toss. If the coin is fair, each of the two values 1 and 0 will be equally likely and thus, at the long run, evenly distributed statistically speaking. We can, however, purposely bend the coin in order to bias the distribution in favor of one of the two possible outcomes. A coin is then a simple model and generator of two value random sequences.

Definition 17 (Memoryless random sequence generator)**.** A memoryless random sequence generator (such as a coin or a die) is a pair $\{\mathcal{A}, \mathrm{P}(x)\}$

where $\mathcal{A}$ is the alphabet or set of letters that can be returned by the generator and $\mathrm{P}(x)$ is the probability function associated with the set.

Example. Let $\mathcal{A} = \{1, 0\}$ and $\mathrm{P}(1) = .25$, $\mathrm{P}(0) = .75$.[5] For this generator 0 has three times the probability of being returned that 1. Thus, after multiple iteration we will find that 1 is generated 25% of the time, while 0 is generated 75% of the time. The following sequence, for example, was generated with these probabilities:

$$\overline{r} = (10000001111000010000000000100000000001011\ldots)$$

Random processes like this are not only simple, they are also rather uninteresting. Music changes; while a random sequence as the one above is locally unpredictable, its global dynamics are static, or, to use the more technical word, *stationary.*

In [3], Crutchfield proposed a curve like the one in figure 4-5 to describe the statistical complexity of processes spanning the whole spectrum between simple periodic processes and complete randomness. The sieve

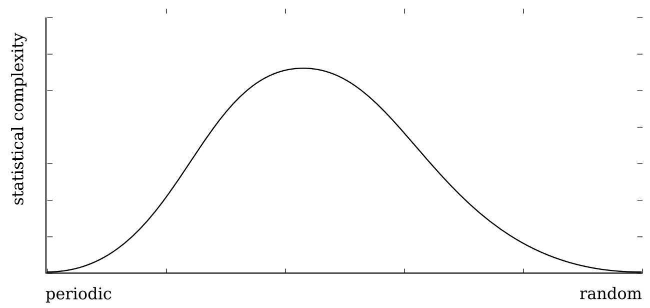


Figure 4-5: Crutchfield's statistical complexity curve.

bases are simple periodic functions and thus fall on the extreme left. The simple random generator given in Definition 17 is on the extreme right. All interesting music is somewhere in between.

[5]Remember that one of the axioms of probability theory is that the sum of the probabilities of all possible mutually exclusive events must add up to one, so $\mathrm{P}(1) + \mathrm{P}(0) = 1$.

In the world of Xenakis sieves, one moves from the extreme left of the graph towards the middle by combining several sieve bases with the boolean operators $(\wedge, \vee, \oplus, \neg)$.

How does one move towards the middle from the right?

Consider the following two sequence fragments:

$$\overline{a} = (1010010001010010100001010010100010010010\ldots)$$
$$\overline{b} = (1011101011101110111110101010111010111010\ldots)$$

Sequence $\overline{a}$ is not as random as $\overline{r}$. In $\overline{a}$, 0 *always* follows 1. Sequence $\overline{b}$ also displays a bit more structure. Notice that, starting from the first 1, every other value is *always* 1:

$$\overline{b} = (\underline{1}0\underline{1}1\underline{1}0\underline{1}0\underline{1}1\underline{1}0\underline{1}1\underline{1}0\underline{1}1\underline{1}1\underline{1}0\underline{1}0\underline{1}0\underline{1}0\underline{1}1\underline{1}0\underline{1}0\underline{1}1\underline{1}0\underline{1}0\ldots)$$

How can the structure of these sequences be characterized for classification and comparison? How can the structure be quantified?

Sequence $\overline{a}$ shows a form of dependency of element $i+1$ on element i. What we can expect from the sequence at time $i+1$ depends to a certain degree of what we have seen at time i; a_i tells us something about what a_{i+1} will be. i.e, the probability $\mathrm{P}(a_{i+1})$ depends on a_i. More formally, $\mathrm{P}(a_{i+1} \mid a_i) \geq \mathrm{P}(a_{i+1})$. Specifically we see that $\mathrm{P}(a_{i+1} = 0 \mid a_i = 1) = 1$, and $\mathrm{P}(a_{i+1} = 1 \mid a_i = 1) = 0$. Because the alphabet $\mathcal{A}$ consists of only the two letters 1 and 0, these two probability measures define a complete probability function conditioned on letter 1. i.e., the probabilities for all the characters in the alphabet (1 and 0) are given and they add up to 1.

The next reasonable thing to do is to compute the probability distribution for the alphabet conditioned on the letter 0. The most straightforward way to do this is by simple count. How many instances of $(0x)$ for all $x \in \mathcal{A} = \{0, 1\}$ are there? There are 12 (00) words and 13 (01) words out of a total of 25 length-2 words starting with 0 in sequence $\overline{a}$. Thus, we can estimate that $\mathrm{P}(0 \mid 0) = \frac{12}{25}$ and $\mathrm{P}(1 \mid 0) = \frac{13}{25}$; each approximately $\frac{1}{2}$. Thus, $\mathrm{P}(0 \mid 0) = \mathrm{P}(1 \mid 0) = 0.5$.

It is clear then that in sequence $\overline{a}$, the prospect of the future given 1 $(\mathrm{P}(x \mid 1))$ is very different from the prospect of the future given 0 $(\mathrm{P}(x \mid 0))$. We can think of these two probability functions as defining two "spaces" or *states*, each having a different "shape of the future". In the context of ϵ-machines (defined below), these probability functions are called *morphs*. Figure 4-6 shows a graphical representation of this.

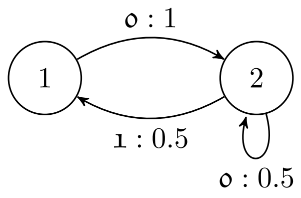


Figure 4-6: A graph representation of the random process generating sequence $\overline{a}$. There are two states, each characterized by having a different morph. State 1 always returns o with probability $\mathrm{P}(\mathsf{o}) = 1$.

Applying the same kind of statistical analysis on sequence $\overline{b}$ results in a profile that is very similar to that of sequence $\overline{a}$ in terms of the estimated probability functions; these are essentially the same, but flipped:

$$\mathrm{P}(x \mid \mathsf{1}) = \begin{cases} 15/28 \approx 0.5, & \text{for } x = \mathsf{1} \\ 13/28 \approx 0.5, & \text{for } x = \mathsf{o} \end{cases}$$
$$\mathrm{P}(x \mid \mathsf{o}) = \begin{cases} 12/12 = 1, & \text{for } x = \mathsf{1} \\ 0/12 = 0, & \text{for } x = \mathsf{o} \end{cases}$$

Sequence $\overline{b}$ is not, however, qualitatively equivalent to the negation of $\overline{a}$. As previously observed, $\overline{b}$ has a clear periodic component absent in $\overline{a}$. The difference between sequences $\overline{a}$ and $\overline{b}$ can be captured clearly in the graphs of the processes. The strictly periodic quality of $\overline{b}$ is encoded in the graph by connecting all outgoing edges of state 2 to state 1, thus creating a regular cycle between the two recovered states. (Figure 4-7). Interestingly, these graph structures can be automatically recovered from the sequences via ϵ-machine reconstruction algorithms, one of which is briefely described in section 4.5.2. While we are at it, notice that $\overline{b}$ can also be modeled as a combination of a simple sieve δ_2^0 and a simple random memoryless process such as $\overline{r}$: $\overline{r} \vee \delta_2^0$.

What is the graph correspoding to sequence $\overline{r}$? The process that generated it is, by definition, memoryless:

$$\mathrm{P}(x) = \begin{cases} .25 & \text{for } x = \mathsf{1} \\ .75 & \text{for } x = \mathsf{o} \end{cases}$$

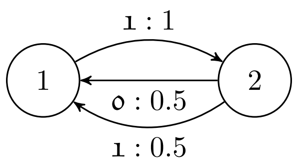


Figure 4-7: A graph representation of the random process derived from sequence $\overline{b}$. The process is strictly periodic, which is reflected in the strict alternation between states. State 1 always outputs 1, state 2 outputs 0 or 1 with equal probabilities.

There is one and only one morph that is constant and unconditional. Thus, the process can be characterized as having a single state with a single morph $\mathrm{P}(x)$. The graph of this process is depicted in Figure 4-8.

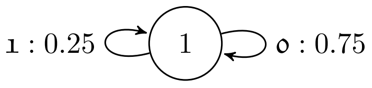


Figure 4-8: The graph structure of the memoryless random process $\overline{r}$.

The graphs of a random process captures the amount of memory it uses and, in a certain way, its complexity.[6]

4.5.1 Computational Mechanics

Given a sequence of values $\overline{s} = s_1, s_2, s_3, \ldots$, is it possible to find a pattern that can explain and compactly represent $\overline{s}$? What causes s_2 to follow s_1? What causes s_3 to follow s_1, s_2? Inversely, having seen s_1, s_2, s_3, what will s_4 most likely be? Or, better yet, what is the distribution of futures given s_1, s_2, s_3? The probably obvious but important implication of the statement "having seen…" is the requirement of memory. If we

[6]A discussion on complexity measures is beyond the scope of this paper. Nevertheless, the reader is encouraged to review Crutchfield and friends' work on the topic ([5, 3, 4, 7, 13]). His *statistical complexity* offers a powerful (and computable!) alternative to the (probably more popular) Kolmogorov-Chaitin complexity. In contrast to the Kolmogorov complexity, which is deterministic and thus maximal when a sequence is random, Crutchfield's statistical complexity is, well, statistical. Besides being computable (and thus practically useful) it corresponds closely (or closer) with our intutition of complexity; in statistical complexity measures, completely random process have complexity 0 and are thus as simple as simple periodic processes. See Figure 4-5.

had no memory about the immediate past we would have difficulties predicting the future. The more we remember, the better our estimate of the future will be. . . or so one would think. But this is not always true. Sometimes retaining more information about the past does not give us a better prediction about the future. In a memoryless random process, such as the toss of a coin, no amount of memorizing past trials will give us a better estimate about the future than a simple guess. So given a particular sequence, how much of it must we remember in order to make the best possible prediction? How can we go about finding the minimum memory required for maximal prediction?

From a statistical point of view, we might do the following: Given a sequence $\overleftarrow{s}$ of length N, first find the alphabet $\mathcal{A}$ of the sequence. Then count how many times each letter appears in the sequence. From the estimate of the relative frequencies of the letters in the alphabet we can make a first informed prediction. Obviously we would predict that the most frequent is the most likely to come next. Can we improve our prediction by keeping track of the past subsequences of length $L = 1$, $\overleftarrow{s}^1$? Count how many times each letter in $\mathcal{A}$ is followed by every other letter. If we can better predict the future with this new calculation , then we continue to $L = 2$ and so on until our prediction can no longer be improved or until we have reached a predefined maximum length limit L_{max}. Note that what we are computing is the probability distribution of the alphabet conditioned on each possible past subsequence $\overleftarrow{s}^L$ of length L. i.e., $\mathrm{P}(a|\overleftarrow{s}^L)$, $\forall\, a \in \mathcal{A}$. In the case of $L = 1$, we have k conditional probability distributions because there are k letters in the alphabet. For $L = 2$ we could potentially have k^2 distinct probability distributions and so on. Some of these distributions, however, could be the same, or practically the same. Thus, it makes sense to group together all subsequences $\overleftarrow{s}^L$ having the same conditional distribution of futures, and to treat them as equivalent given that they have the same "shape" and predictive power. i.e., there is no need to distinguish between different past sequences that give us the exact same information about the future. Doing so would simply add more memory requirements without giving us any added predictive power.[7] At the end of the analysis, having reached an L_{max}, we end up with a collection Σ of sets σ of sequences $\overleftarrow{s}^L$ grouped together because they share the same probability distribution of futures.

What we have just done is effectively partitioned the space of all past subsequences $\overleftarrow{s}^L$ on the basis of their probability distribution of futures. This set Σ gives us an interesting first characterization of the process that might have generated the sequence observed. The next stage is to see how these σ sets relate to each other; i.e., how they connect and how the process jumps from one set to the next.

Some notation: $\overline{S}$ is a sequence of random variables $\overline{S} = \ldots, S_{i-1}, S_i, S_{i+1}, \ldots$. S_i is the i^{th} random variable of the sequence. s_i is the actual value returned by S_i. $\overrightarrow{S_i}^L$ is the sequence $S_i, S_{i+1}, \ldots, S_{i+L-1}$ of L random variables. $\overleftarrow{S_i}^L$ is the sequence $S_{i-L}, \ldots, S_{i-2}, S_{i-1}$ of L random variables immediately preceding S_i.

Definition 18 (Stochastic Process)**.** Let $\mathcal{A}$ be a countable set, $\Omega = \mathcal{A}^{\mathbb{Z}}$ be the set of sequences composed from $\mathcal{A}$, and P the probability measure associated with set Ω. A process is a sequence $\overline{S}$ of random variables S_i with values drawn from the set $\mathcal{A}$.[7]

4.5.2 ϵ-machines

An ϵ-machine is a computational model reconstructed from some data stream.

Definition 19 (ϵ-machine)**.** An ϵ-machine of a process is the pair $\{\Sigma, \mathbf{T}\}$, where Σ is the set of *causal states*, and $\mathbf{T}$ is the set of labeled transition matrices $T_{ij}^{(s)}$. ϵ-machines can be visualized as graphs, where vertices (nodes) are the *causal states*, and the edges are the labeled transition probabilities.

Definition 20 (Causal State)**.** The *causal states* of a process are the equivalence classes σ_i induced by the equivalence relation $\sim_\epsilon$ defined as:

$$\overleftarrow{s_i} \sim_\epsilon \overleftarrow{s_j} \text{ iff } \mathrm{P}(\overrightarrow{s}|\overleftarrow{s_i}) = \mathrm{P}(\overrightarrow{s}|\overleftarrow{s_j}) + \delta \quad \forall \overrightarrow{s},$$

where δ is a tolerance.[3] We write σ_i the i^{th} *causal state*, Σ the set of all causal states, and $\mathcal{S}$ the corresponding random variable.[8]

Each *causal state* has the following attributes:

[7]Refer to [12] for a more rigorous and complete definition.

[8]A couple of equivalent definitions have been given by different authors. See, for example, [12] and [7].

1. An index i, or "name".
2. A set of histories $\{\overleftarrow{s} \in \sigma_i\}$.
3. A conditional distribution over futures: $\mathrm{P}(\overrightarrow{S}|\sigma_i) = \mathrm{P}(\overrightarrow{S}|\overleftarrow{s}), \overleftarrow{s} \in \sigma_i$, called the *morph*[5].

The *morph* is the probability distribution of futures of a given *causal state* σ. Subsequences $\overleftarrow{s_i}$ are grouped together in the same *causal state* precisely because they share the same *morph*.

Definition 21 (Causal Transitions)**.** The labeled transition probability $T_{ij}^{(s)}$ is the probability of making the stransition from state σ_i to state σ_j while emitting the symbol (letter) $s \in \mathcal{A}$:

$$T_{ij}^{(s)} \triangleq \mathrm{P}(\mathcal{S}' = \sigma_j, \overrightarrow{S}^1 = s|\mathcal{S} = \sigma_i),$$

where $\mathcal{S}$ is the current *causal state* and $\mathcal{S}'$ its successor. $\mathbf{T}$ is then the set $\{T_{ij}^{(s)} : s \in \mathcal{A}\}$.[12]

ϵ-machine reconstruction

The "intuitive" algorithm described above is a rough analogue of the first stage in the ϵ-machine reconstruction algorithm proposed by Crutchfield and Young.[3] In [12] and [13], Shalizi proposed an improved algorithm and an implementation. This ϵ-machine reconstruction algorithm tries to infer the minimal Markovian model capable of generating a given sequence. It begins by assuming that the sequence to be analyzed comes from a simple memoryless random process having a single *causal state*. i.e., it assumes the simplest possible process as point of departure for the reconstruction. Broadly, the algorithm is as follows:[9]

1. *Homogeneity.* Compute the causal states whose member "words" (subsequences) have the same *morph*.

   (a) For each $\hat{\sigma} \in \Sigma$, compute its *morph*:

      i. For each sequence $\overleftarrow{s}^L \in \hat{\sigma}$, compute its *morph*.

      ii. Average the *morphs* of the sequences in $\hat{\sigma}$ to obtain the *morph* of $\hat{\sigma}$.

[9]Please refer to [13] for a detailed definition, explanation and discussion.

   (b) For each $\hat{\sigma} \in \Sigma$, test the null hypothesis.Find the *causal state* of suffix sequence $a\overleftarrow{s}^L$ with length $L+1$, for each length L subsequence $\overleftarrow{s}^L \in \hat{\sigma}$ and each $a \in \mathcal{A}$.
      i. Compute the *morph* of $a\overleftarrow{s}^L$.
      ii. See if this *morph* matches the *morph* of a previously obtained state.
      iii. If it does, add it to the matching state, otherwise create a new state and add the sequence $a\overleftarrow{s}^L$ to it.

2. *Determinization.* We now compute the state transitions to connect the states, and we make sure every member (subsequence) of each state $\hat{\sigma}_i$ has the same *successor state* for the same symbol $a \in \mathcal{A}$. i.e., we determinize them.

   For each state $\hat{\sigma} \in \Sigma$:

   (a) For each $a \in \mathcal{A}$:
      i. For all $\overleftarrow{s} \in \hat{\sigma}$ find the successor state on a. i.e., Find the state $\hat{\sigma}'$ that has the same morph as $\overleftarrow{s}\, a$, for all $\overleftarrow{s}$.
      ii. If there is only one successor state on a, go to the next a.
      iii. If there are $n \geq 2$ successor states on a, create $n-1$ new states and move histories from $\hat{\sigma}$ to the new states so that each state has the same successor on a.

   (b) If every history $\overleftarrow{s} \in \hat{\sigma}$ has the same successor on a, for every a, go to the next state.

CHAPTER FIVE
1-bit Composition

To compose is to construct large "complex" perceptually non-trivial structures from small simple parts.

5.1 Parallel and Sequential Composition

There are two basic ways in which two sequences $\overline{x}$ and $\overline{y}$ can be composed: in *parallel* and in *sequence.*

In sequential composition, one sequence is placed before the other. Sequencing is achieved with the concatenation operator '$|$'. If $\overline{x} = (10010)$ and $\overline{y} = (101010)$, for example, then $\overline{x} \mid \overline{y} = (10010101010)$.

Parallel composition is a bit more interesting. In its simplest form, two sequences $\overline{x}$ and $\overline{y}$ are composed in parallel when they are superimposed, occurring simultaneously. i.e., they overlap in time. We can define a parallel operator '$\|$' that superimposes two or more sequences to compose a set of parallel sequences.

Definition 22 (Parallel operator '$\|$')**.** Let x and y be either boolean vectors or sets of boolean vectors. Then $x \parallel y$ yields a set of boolean vectors. If $\overline{x} = (x_1, x_2, \ldots, x_m)$ and $\overline{y} = (y_1, y_2, \ldots, y_m)$ are two boolean vectors, then the composition of $\overline{x}$ and $\overline{y}$ via the $\|$ operator yields an unordered set $\overline{\mathbf{z}}$ of vectors $\overline{x}$ and $\overline{y}$:

$$\overline{\mathbf{z}} = \overline{x} \parallel \overline{y} = \left\| \begin{matrix} (x_1, x_2, \cdots, x_m) \\ (y_1, y_2, \cdots, y_m) \end{matrix} \right\| .$$

If $\overline{x} = (x_1, x_2, \ldots, x_m)$ and $\overline{\mathbf{z}} = \left\| \begin{matrix} (a_1, a_2, \cdots, a_m) \\ (b_1, b_2, \cdots, b_m) \end{matrix} \right\|$ is a set of two boolean vectors $\overline{a}$ and $\overline{b}$, then the composition of $\overline{x}$ and $\overline{\mathbf{z}}$ via the $\|$ operator yields the unordered set $\overline{\mathbf{w}}$ of vectors $\overline{x}$, $\overline{a}$, and $\overline{b}$:

$$\overline{\mathbf{w}} = \overline{x} \parallel \overline{\mathbf{z}} = \left\| \begin{matrix} (x_1, x_2, \cdots, x_m) \\ (a_1, a_2, \cdots, a_m) \\ (b_1, b_2, \cdots, b_m) \end{matrix} \right\|.$$

Note that the vertical arrangement between vectors in $\overline{\mathbf{z}}$ or $\overline{\mathbf{w}}$ is immaterial, and is only used to visually suggest the simultaneity.

Another form of parallel composition is through *combinations*. We have already encountered this form of parallel composition in the use of binary operators. The combination of two one-dimensional sequence $\overline{x}$ and $\overline{y}$ via the binary operators $\vee$, $\wedge$ and $\oplus$ is such a type of parallel composition. Note that superposition (via '$\|$') and combination are quite different in terms of the final structure. While combinations collapse the combined sequences into a new one-dimensional sequence, '$\|$' simply "runs them together". This two-dimensional structure can be an intermediate step for further processing before collapse, or it can be the desired final structure. This will be the case when, for example, more than one "instrument" plays simultaneously, each executing a different row in the matrix.

Two one-dimensional elements $\overline{x}$ and $\overline{y}$ can be constructed *independently* before a composition $\overline{x} \parallel \overline{y}$. Alternatively $\overline{x}$ and $\overline{y}$ can be conceived together as part of a single multidimensional entity $\overline{\mathbf{z}} = \overline{x} \parallel \overline{y}$ in which the vertical configurations between the elements of $\overline{x}$ and $\overline{y}$ are accounted for. This is 1-bit counterpoint. In contrapuntal writing two or more concurrent streams are composed such that the vertical relations are considered and thus coupled to some degree. The process of simply superimposing two independent sequences with the '$\|$' operator can be called *layering*, while the multidimensional contrapuntal composition, *braiding*. The parallel sequences composing a multidimensional braid can, however, have multiple degrees of interdependence. Thus, a continuous spectrum exists between total independence and complete coupling for two parallel sequences.[1]

[1]A discussion on coupling measures is beyond the scope of this paper. From an information theoretic stance though, the *mutual information* measure $I[X;Y] =$

A clear example of tight contrapuntal coupling can be found in the first measures of Xenakis' *Psappha* for solo percussion (Figure 5-1). In this

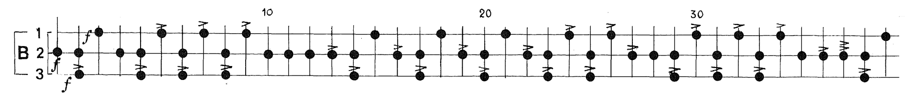


Figure 5-1: The opening measures of Xenakis' *Psappha* for percussion. Black dots indicate a stroke. Layer B in the score is divided into three sub-layers numbered 1, 2, 3, each corresponding to a different instrument. Layers 1 and 2 are perfectly coupled as one is the complement of the other.

passage, sub-layers 1 and 2 are perfectly coupled because they are perfect complements of each other: $B_2 = \neg B_1$.[2] i.e., sublayer 1 tells us everything there is to know about layer 2 and vice versa. Note also that sub-layer 1 (B_1) is identical to B_3 but delayed by 1 step. Thus, all three layers are coupled, each pair in varying degrees. This coupling would have hardly occurred had they been composed independently, with no knowledge of each other. Either two lines are composed based on one original "source" line (this is actually what Xenakis did) —we might call this *horizontal braiding*—, or all three are composed as a three-dimensional sequence —*vertical braiding.* The first approach might go something like this: First define $B_2 = (1101101010111110110110110101110101011110)^3$; then define $B_1 = \neg B_2$; then $B_3[n] = B_1[n+1]$ for all n; finally $\mathbf{B} = B_1 \parallel B_2 \parallel B_3$.

For the second method (vertical braiding) we take each vertical configuration as a"character" in our alphabet. Thus we move from $\mathcal{A}$ to $\mathcal{A}^3$. There are 2^3 possible words in this alphabet ($\substack{\cdot\\\cdot\\\cdot}$, $\substack{\bullet\\\cdot\\\cdot}$, $\substack{\cdot\\\bullet\\\cdot}$, $\substack{\cdot\\\cdot\\\bullet}$, $\substack{\bullet\\\bullet\\\cdot}$, $\substack{\bullet\\\cdot\\\bullet}$, $\substack{\cdot\\\bullet\\\bullet}$, $\substack{\bullet\\\bullet\\\bullet}$), but in the Xenakis fragments only three are used: $\substack{\bullet\\\cdot\\\cdot}$, $\substack{\cdot\\\bullet\\\cdot}$ and $\substack{\cdot\\\bullet\\\bullet}$. Thus, $\mathbf{B}$ can be composed by concatenating the three characters: let $\mathbf{b}_1 = \substack{\bullet\\\cdot\\\cdot}$, $\mathbf{b}_2 = \substack{\cdot\\\bullet\\\cdot}$, and $\mathbf{b}_3 = \substack{\cdot\\\bullet\\\bullet}$, then $\mathbf{B} = \mathbf{b}_2 \mid \mathbf{b}_3 \mid \mathbf{b}_1 \mid \mathbf{b}_2 \mid \mathbf{b}_3 \mid \mathbf{b}_1 \mid \mathbf{b}_3 \mid \mathbf{b}_1 \mid \cdots$.

$H[X] - H[X \mid Y]$ is a good place to start. In particular, Kraskov et al.[10] define a normalized mutual information metric $D(X,Y) = 1 - \frac{I[X;Y]}{H[X,Y]}$, which I have used in [1].

[2]Refer to [1] for a detailed analysis of the passage.

[3]Xenakis defines B_2 with the following sieve expression: $[(\delta_8^0 \vee \delta_8^1 \vee \delta_8^7) \wedge (\delta_5^1 \vee \delta_5^3)] \vee [(\delta_8^0 \vee \delta_8^1 \vee \delta_8^2) \wedge \delta_5^0] \vee [\delta_8^3 \wedge (\delta_5^0 \vee \delta_5^1 \vee \delta_5^2 \vee \delta_5^3 \vee \delta_5^4)] \vee [\delta_8^4 \wedge (\delta_5^0 \vee \delta_5^1 \vee \delta_5^2 \vee \delta_5^3 \vee \delta_5^4)] \vee [(\delta_8^5 \vee \delta_8^6) \wedge (\delta_5^2 \vee \delta_5^3 \vee \delta_5^4)] \vee (\delta_8^1 \wedge \delta_5^2) \vee (\delta_8^6 \wedge \delta_5^1)$.

While the final sequence is the same for both, the structural composition is entirely different: the first is a parallel of sequences, the second a sequence of parallels.

5.1.1 Parallel (simultaneous) Perceptual Streams

Suppose we have two regular impulse sequences $\overline{x} = (800)_{\Delta\circlearrowleft}$ and $\overline{y} = (500)_{\Delta\circlearrowleft}$ with a tatum of $T = 0.001$ sec. For $\overline{y}$, a pulse will be heard every $500/1000 = 1/2$ second; for $\overline{x}$ every $800/1000 = 4/5$ second. We want both to be perceived as parallel independent streams, but simply combining the two with an OR operator ($\overline{x} \vee \overline{y}$) will not work. The mix will dissolve the regular pattern of each sequence and we will fail to perceive the two streams we originally intended to carry across. This is because the impulses in the two sequences are identical and thus indistinguishable from each other. They must be made distinct to be perceptually separable.

In order to hear multiple parallel and perceptually independent streams, in traditional instrumental writing each of the sequences is assigned to a different instrument or "voice". This works because, from a psychoacoustic perspective, each instrument has its own distinct timbre. From a purely acoustic perspective (and very broadly speaking), we can say that each instrument has it's own "fingerprint" response to a particular stimulus. For example: a high-hat and a bass drum can be excited by hitting them in the exact same way with the same mallet, and yet they will sound very different. The two instruments respond differently to the same stroke; thus we can say that they have a different "stroke response". The stroke is not unlike an *impulse signal* in that it is very short, almost perceptually instantaneous. Thus, the term *impulse response* is the name given to the output of any system given an impulse as input.[4]

The same principle can be applied to 1-bit music. A short, high frequency impulse sequence is driven by a low frequency impulse train. This operation can be perform by convolving a control signal $\overline{x}$ with a "voice" or "instrument" signal $\overline{h}$.

Definition 23 (Convolution)**.** The convolution $(f * g)$ between two real functions f and g is defined as the integral of the product of the two

[4]The impulse response is also called the *convolution kernel*[15].

functions, where one is reversed and shifted:

$$(f * g)(t) \triangleq \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

Here, we are interested in a discrete time boolean version of this.

Definition 24 (Boolean OR Convolution)**.** Let $\overline{x}$ be an N point boolean vector and $\overline{h}$ an M point boolean vector. The convolution $\overline{x} \overset{\vee}{*} \overline{h}$, is an $N+M-1$ point sequence defined as

$$(\overline{x} \overset{\vee}{*} \overline{h})[n] \triangleq \bigvee_{j=0}^{M-1} \overline{x}[n-j] \wedge \overline{h}[j]$$

Definition 25 (Boolean XOR Convolution)**.**

$$(\overline{x} \overset{\oplus}{*} \overline{h})[n] \triangleq \bigoplus_{j=0}^{M-1} \overline{x}[n-j] \wedge \overline{h}[j]$$

Example (OR convolution). Let $\overline{x} = (100000100000)$, $\overline{h} = (11)$, and $\overline{k} = (111)$, then

$$\overline{x} \overset{\vee}{*} \overline{h} = (110000110000)$$
$$\overline{x} \overset{\vee}{*} \overline{k} = (111000111000).$$

This minimal example shows how a slow moving pattern can be made to have different "characters" by convolving it with different, short, high rate sequences. In a sense, we can think of the convolution between two boolean signals as one driving the other. i.e., each impulse of the control sequence $\overline{x}$ triggers a complete sequence $\overline{h}$ or $\overline{k}$.

The separation between the control signal $\overline{x}$ and the high rate "fingerprints" $\overline{h}$ and $\overline{k}$ is convenient because it allows us to reuse and replace one or the other in different situations. Here again we encounter a case where the separation of parameters is a powerful principle.

5.2 Modulation and Phasors

A process modulates another if the former governs at least one parameter of the later. Probably the most well known forms of modulation are AM (amplitude modulation) and FM (frequency modulation) commonly applied to sinusoids in sound synthesis and radio broadcast. Here I discuss modulation in the context of phasors.

Consider the following acceleration sequence:

$$\overline{s} = (\mathtt{110101000100100000100010000000100100000101000110})$$

In delta notation: $\overline{s}_\Delta = (1, 2, 2, 4, 3, 6, 4, 8, 3, 6, 2, 4, 1, 2)_\Delta$. How might we construct this sequences? A sieve XOR decomposition of $\overline{s}$ gives $\mathrm{rc}(30, 0)$, $\mathrm{rc}(32, 1)$, $\mathrm{rc}(19, 3)$, $\mathrm{rc}(34, 5)$, $\mathrm{rc}(36, 9)$, $\mathrm{rc}(34, 12)$, $\mathrm{rc}(30, 18)$. Not very intuitive in terms of a way of constructing the sequence $\overline{s}$. There is hardly a clear connection between the sieve basis function and $\overline{s}$. In modeling a sequence we want a compact representation and a practical tool having few and meaningful parameters. Note that the sequence has a pattern; in the inter-impulse-interval sequence, even indexes (counting from 0) are multiples of 1, odd indexes are multiples of 2:

$$(1, \square, 2, \square, 3, \square, 4, \square, 3, \square, 2, \square, 1)_\Delta$$
$$(\square, 2, \square, 4, \square, 6, \square, 8, \square, 6, \square, 4, \square, 2)_\Delta$$

Taking the sequence as a succession of pairs we see that the ratio between numbers in each pair is the same, $1 : 2$, repeating several times at varying scales, as if a ratchet was being played progressively slower and then faster. Thus, the sequence can be constructed as $1 \cdot (1, 2)_\Delta \mid 2 \cdot (1, 2)_\Delta \mid 3 \cdot (1, 2)_\Delta \mid 4 \cdot (1, 2)_\Delta \mid 3 \cdot (1, 2)_\Delta \mid 2 \cdot (1, 2)_\Delta \mid 1 \cdot (1, 2)_\Delta$.

In Section 4.3 I discuss phasors as cyclical patterns that are unique in terms of their inter-impulse-interval configurations. i.e., they define an equivalence class of boolean cycles on the basis of their inter-impulse-intervals. $\overline{s}$ can then be modeled simply as a phasor $\overline{x}^0_\phi = (\mathtt{110})^0_\phi$ rotated at different speeds. For this a function $f : \mathbb{Z} \to \mathbb{B}$ can be defined as follows:

$$f(p_n) = \begin{cases} \overline{x}^0_\phi[p_n], & \text{if } d(p_n) \neq 0 \\ \mathtt{0}, & \text{otherwise,} \end{cases}$$

where $d(p_n)$ is the difference of p_n: $p_n - p_{n-1}$. The function takes a phase step p and returns '$\mathtt{1}$' or '$\mathtt{0}$'.

Then $\overline{s} = (f(\overline{p}_i))$ for all $\overline{p}_i$ in the phase sequence

$\overline{p} = (0, 1, 2, 3, 3, 4, 4, 5, 5, 6, 6, 6, 7, 7, 7, 8, 8, 8, \cdots, 15, 15, 16, 16, 17, 17, 18, 19, 20).$

This model also provides a simple way of adding some irregularity to arbitrary periodic sequences. e.g., drive $f(\cdot)$ with $\overline{p} + \overline{r}$, where $\overline{r}$ is some form of noise.

5.3 Hierarchic stochastic finite state machines

In the previous chapter ϵ-machines were introduced and an algorithm for automatic ϵ-machine reconstruction was described. These finite state automata can be constructed directly for the purpose of composing sequences. Naturally, the characteristics of these sequences will be determined by the graph structure of the machine and the probabilities associated with each edge in the graph.

While each state has a distinct probability map associated with the alphabet $\mathcal{A}$, the overall machine as a whole is stationary in the sense that these probability functions aren't expected to change. After the output of the machine reaches a certain length, the generated sequence will begin to feel stationary. In order to overcome this, the machine needs to grow larger, with more states and more connections between states. Unfortunately, the complexity of these machines can grow to become quite unmanageable quickly. There are, however, two practical and intuitive ways to manage this. In both alternatives, instead of constructing a single, large finite state machine with many states and edges from the outset, we can create various small machines that can then:

1. be connected to each other at one or a few places (Figure 5-2).
2. be hierarchically nested, so that one machines activates other machines, which in turn activate other machines, . . . , which in turn output characters from the alphabet of interest: $\mathcal{A}$ (Figure 5-3). In this case, an additional *null* state must be added to all but the outermost machine to trigger their termination, allowing the traversal of the machine hierarchy.

A hierarchical machine is not unlike what one normally finds in many musical structures. e.g., motives, which are structured into themes, which are structured into phrases, which are structured into sections. The difference with hierarchical finite state machines is that these are

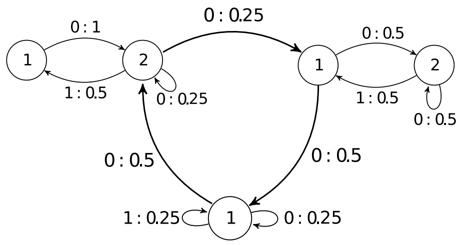


Figure 5-2: Example of three independently constructed ϵ-machines connected together to form a single larger ϵ-machine.

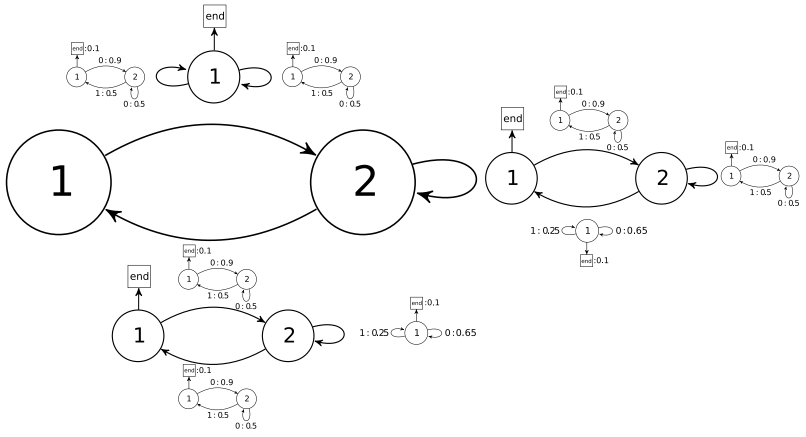


Figure 5-3: Example of a thee-level hierarchy of two and one state ϵ-machines.

stochastic at all levels, not deterministic. However, since the probabilities associated with the edges can be arbitrarily defined, anything in the spectrum between uniform randomness and complete determinism can be constructed.

5.4 Fonoptera and the Tepozchiquilichtli Studies

The models and methods described in this paper have been implemented in sofware and used to compose a series of 1-bit music studies titled *Tepozchiquilichtli*[5] and a piece for eight 1-bit instruments and four MIDI player pianos that stems from these studies, titled *Fonoptera*. In what

[5]Tepozchiquilichtli: (Nahuatl to Spanish transliteration) *tepoz* metal + *chiquilichtli* cicada.

follows I will give a brief description of the use of these tools in some of the *Tepozchiquilichtli* studies and *Fonoptera*.

Tepozchiquilichtli I

Tepozchiquilichtli I is comprised of two section. The first is composed of seven concurrent periodic sequences whose frequencies varies constantly. The opening of the piece is upward moving *glissandi* that mimic the song of some cicadas. These are implemented as simple $(\imath)_\phi$ phasors. Each of the multiple phasors is then convolved with a different high rate convolution kernel (as described in section 5.1.1) to make them all perceptibly distinct from one another.

Just as ASCII art approximates high resolution photographs with the limited ASCII character set, the second section of *Tepozchiquilichtli I* is based on the idea of making the best frequency domain 1-bit approximation of 16-bit recordings. The reconstruction alphabet in this case is the set of all boolean cycles $\overline{c}_{\circlearrowleft_i}$ up to an arbitrary maximum period. The algorithm for finding the sequence of irreducible boolean cycles $(\overline{c}_{\circlearrowleft_a}, \overline{c}_{\circlearrowleft_b}, \overline{c}_{\circlearrowleft_c}, \cdots)$ that is spectrally closest to the original 16-bit sequence $\overline{s}$ is the following:

1. Take the STFT (Short Time Fourier Transform) $\mathbf{S}$ of an audio recording $\overline{s}$.
2. Take the Fourier Transform C_i of each and every boolean cycle $\overline{c}_{\circlearrowleft_i}$ (see Section 4.2) up to an arbitrary maximum period.
3. For each STFT slice $\mathbf{S}_j$, find the C_i most similar to it: $\arg\max_i \langle \mathbf{S}_j, C_i \rangle$, where $\langle \mathbf{S}_j, C_i \rangle = \sum_k \mathbf{S}_j[k] C_i[k]$.
4. Create a new sequence $\overline{z}$ composed by concatenating all the boolean cycles that were found closest to $\mathbf{S}_j$ for all j.

Tepozchiquilichtli II

The composition of *Tepozchiquilichtli II* is the simplest and most compact of studies I-VI; it is, actually, trivial. Nevertheless it demonstrates one of the nice properties of sieves, which is that long continuously changing sequences (and thus, memory hungry from the point of view of ϵ-machines) can be easily created. The study is composed of an introduction, which is a gradual acceleration of phasor $(\imath)_\phi$, followed by the following sieve expression:

$$\bigvee_{i=0}^{14} \mathrm{rc}(293 + i, 19 \cdot i)$$

The sieve defines a set of displaced poly-rhythms with ratios $293 : 294 : \cdots : 307$. Combined, they form a single, slowly moving sequence of pulses that seem to flow in and out.

Tepozchiquilichtli III

The main musical construct in *Tepozchiquilichtli III* is a parallel combination of phasors $(\mathtt{1})_\phi$ all driven at the same rate (i.e., having the same period) but with slightly different amounts of jitter, causing the phasors to gradually move out of phase relative to each other. At frequencies within the pitch range, this construct creates a sound with a fixed pitch but varying timbre. The number of phasors staked together and the amount of jitter are the two parameters that control the timbre. The greater the jitter, the noisier the sequence. Thus, this study explores variations of timbre around a single pitch, and the spectrum between pitched sounds and noise.

Tepozchiquilichtli VI

From a perceptual perspective, this is a study on perceived rhythm and timbre, and the fuzzy in between. Structurally, it is a study on simple two-state stochastic finite automata like the ones discussed in Section 4.5. If we limit the universe of two-state machines to those where each state has two edges (i.e., those machines where each state has exactly the same number of edges as there are states in the machine), then there are only the three possibilities depicted in Figure 5-4.

The three machines will generate different kinds of sequences due to their different graph structures, but the probabilities associated with the edges also determine the quality of the sequences generated. Because there are two states, there are two morphs $\mathrm{P}(a|\sigma_1)$, and $\mathrm{P}(a|\sigma_2)$, and four probabilities total, two per state: $p = \mathrm{P}(a|\sigma_1)$, $1 - p = \mathrm{P}(\neg a|\sigma_1)$, $q = \mathrm{P}(b|\sigma_2)$, and $1 - q = \mathrm{P}(\neg b|\sigma_2)$. These four probabilities can be assigned to the edges in four different ways. Thus, one and the same two-state machine graph can generate sequences with different qualities depending on how the probabilities are assigned to the edges.

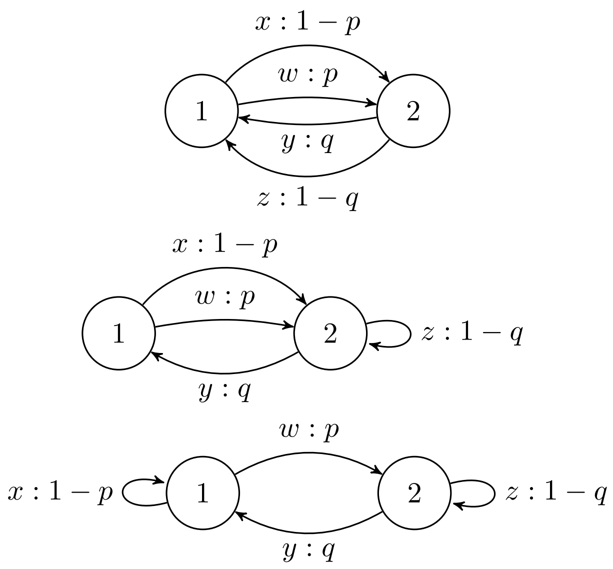


Figure 5-4: The three possible two-state machines with two edges per state.

Tepozchiquilichtli VI is entirely composed from a single stochastic finite state machine with eight states (Figure 5-5). The machine is constructed out of four interconnected two-state sub-machine "kernels"; specifically, the third two-state machine in Figure 5-4 with the two self loops. The difference between the four two-state kernel machines is in the way the four probabilities p, $1 - p$, q, and $1 - q$ are assigned to the four edges of each machine.

The alphabet of this machine is composed not of isolated boolean values 1 and 0, but of short boolean vectors. Thus, each edge of the machine throws a boolean sequence rather than just a single boolean symbol. The whole machine takes eight sequences labeled $\overline{w}$, $\overline{x}$, $\overline{y}$, $\overline{z}$, $\overline{a}$, $\overline{b}$, $\overline{c}$, $\overline{d}$. The first four are used in each and every two-state sub-machine; the last four are for the edges connecting the four two-state sub-machines. While the machine remains unchanged throughout the whole piece, the definition of the alphabet (the vectors associated with each edge) changes. For example, for one section $\overline{w} = (01)$, $\overline{x} = (01)$, $\overline{y} = (10)$, $\overline{z} = (010)$, while for another $\overline{w} = (100)$, $\overline{x} = (0)$, $\overline{y} = (110)$, $\overline{z} = (01)$.

Fonoptera

Fonoptera: from Greek, *fono* sound + *pteron* wing; read *sounding insect.*

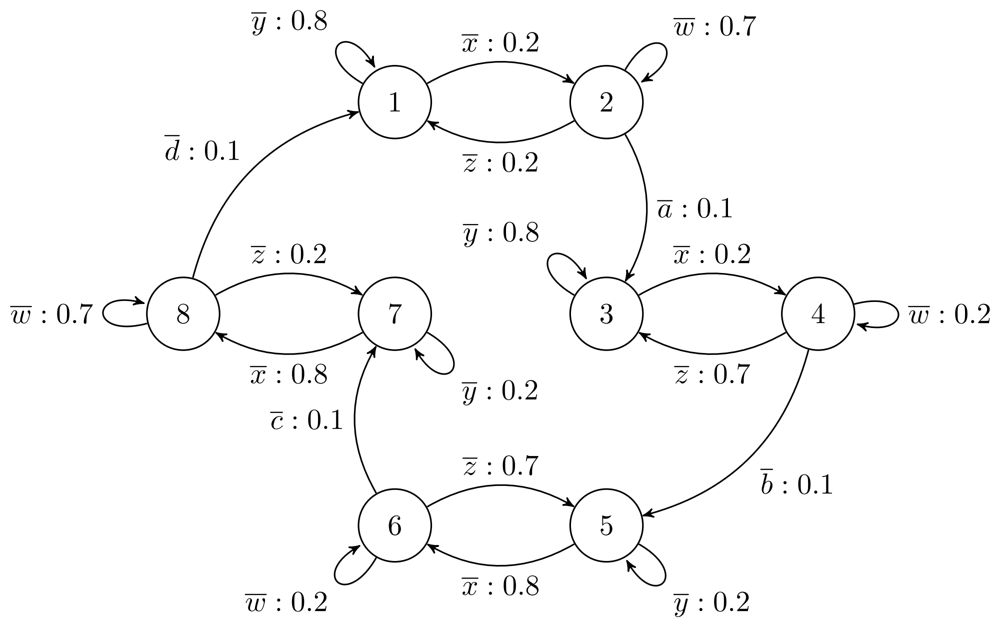


Figure 5-5: The eight state machine used to construct *Tepozchiquilichtli VI*.

Fonoptera represents the culmination of a period of my own research into the world of 1-bit music. It is written for eight custom-made 1-bit instruments (see Appendix B) and four mute MIDI pianos. Because the piece is about 1-bit music, the pianos are brought closer to the sound world of my custom made 1-bit instruments by damping their strings to avoid the production of clear pitches and by using the same dynamic level for every stroke. All twelve instruments are controlled and synchronized with a single computer.

Fonoptera is composed of seven sections, five of them deriving from the *Tepozchiquilichtli* studies. The first section is an introduction that consists of a gradual de-phasing of two regular pulses. This introduction is only played by the 1-bit instruments. The second section is built on *Tepozchiquilichtli II*, and is the superposition of eight simple regular pulses with periods 410, 411, 412, 413, 414, 415, 416, 417 and 418; it is played only by piano 1, and serves as a stable context from which chaos will later emerge. The third section marks a sudden jump from simple regular structures to randomized sequences. It is characterized by the superposition of three parallel stochastic machines, each running at a different rate: tatums 800 ms, 83 ms and 25 ms. Thus the same machine is run (and heard) below and above the rate of pitch perception. In addition, each is executed with a different instrument/timbre: the low strings of the piano, the high strings of the piano, and the 1-bit cicada instruments. Section four derives from *Tepozchiquilichtli IV* and is characterized by the recurrence of an acceleration gesture that starts at approximately 12 Hz (the sub-pitch range) and ends in frequencies of around 130 Hz. Section five is derived from *Tepozchiquilichtli VI*, and marks a return to irregular rhythms. As in section 3, in this section a single stochastic machine is run at different rates. Here, however, the sequences are arranged sequentially rather than in parallel; thus, the listener is made to experience different percepts—from the same kinds of sequences—due to the difference in playback rates. Section six is based on *Tepozchiquilichtli III* and is executed by the 1-bit cicada instruments exclusively. It is the loudest and most intense section of the piece; it is the insects' desperate call for the rain, and a sign of their demise. The final section, the rain, is a wash of irregular impulses whose mean frequency and variance change constantly and continuously. The section is derived from *Tepozchiquilichtli V*, and is executed by both the pianos and the 1-bit cicada instruments.

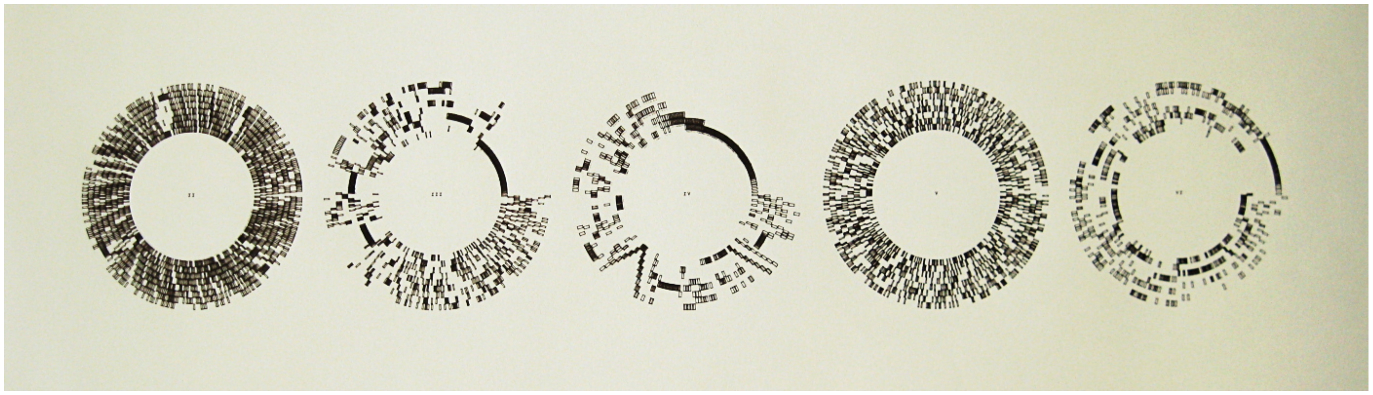


Figure 5-6: Radial scores for *Tepozchiquilichtli II*, *Tepozchiquilichtli III*, *Tepozchiquilichtli IV*, *Tepozchiquilichtli V* and *Tepozchiquilichtli VI*. The 360 degrees of the circle represents the totality of the duration of each study. Each impulse is represented with a rectangular 'tick' around the circle. The position of each tick along the radius (the distance from the center) represents the instrument number executing the impulse.

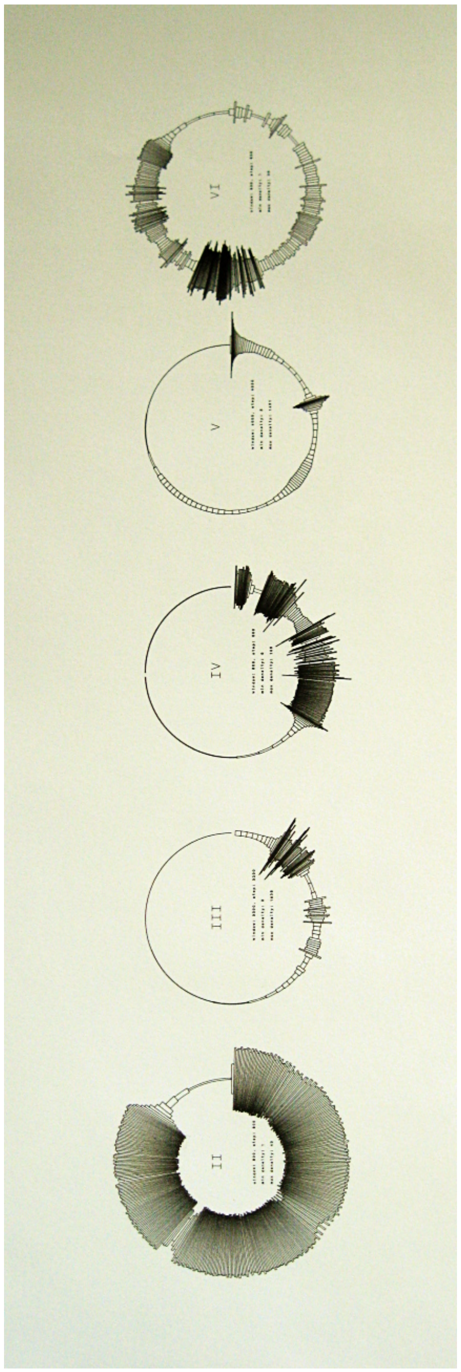


Figure 5-7: Radial scores for *Tepozchiquilichtli II*, *Tepozchiquilichtli III*, *Tepozchiquilichtli IV*, *Tepozchiquilichtli V* and *Tepozchiquilichtli VI*. The 360 degrees of the circle represents the totality of the duration of each study. Each radial rectangle represents the density of events in the study at a given time-frame/window. The greater the density of events, the taller and thinner the rectangle; the lower the density, the wider and shorter the rectangle.

Appendix A Notation

$\mathbb{N}$	Set of natural numbers including 0.
$\mathbb{N}^*$	Set of natural numbers excluding 0.
$\mathbb{B}$	Boolean set $\{\mathtt{1}, \mathtt{0}\}$.
x	Scalar variable.
$\overline{x}$	Vector.
$\overline{\mathbf{x}}$	Two-dimensional matrix.
$\overline{x}[n]$	Sequence indexing. Returns the n the element or $\overline{x}$.
x_i, s_i	Similar to $\overline{x}[n]$, this is the i^{th} element of $\overline{x}$ (or $\overline{s}$).
$\sim_{\circlearrowleft}$	Cyclically equivalent.
$\langle a, b\rangle$	Inner product of a and b.
$\wedge$	Logical AND.
$\vee$	Logical OR.
$\oplus$	Logical XOR.
$\neg$	Logical NOT.
$\mathrm{P}(a)$	Probability of a.
$\mathrm{P}(a \mid b)$	Probability of a given b.
S	A random variable.

Appendix B
1-bit Instrument Construction

In looking for a truly 1-bit mechanical device I started to experiment with dot matrix printers. The print head of these machines is an array of tiny hammers. These hammers push the ink against the paper, following the exact same mechanism as the even older mechanical typewriters dating back to the XIXth century. After some iterations of hacking dot matrix printers—keeping those parts I actually needed and removing those I did not—I ended up with the construction shown in Figure B-1. Each of these 1-bit instruments is made of one dot matrix print head mounted on a wooden base and a plastic cap held floating in front of the print head with three springs. When the print head triggers, its little hammers strike the plastic cap projecting the sound forward.[1]

Controlling the Instruments

A dot matrix print head is an array of tiny solenoids (i.e., electrical hammers). Each of the solenoids in the head is activated with a 35 volt impulse lasting no more than 100 μs (holding the high voltage much longer will burn the solenoids). I have used the Arduino[2] micro-controller platform to control several print heads simultaneously. The Arduino board, however, operates on 5 volts. In order to drive the dot matrix print heads with the micro-controller, an intermediary driver is required; the function of this driver is to send the appropriate 35 volts to the print head upon receiving a 5 volt signal from the Arduino. I designed and built a simple dot matrix print head driver for this purpose. The heart

[1]The insect wings that are carved into the wooden base are not, as most people think, butterfly wings, but open cicada wings. The sound of these instruments always reminds me of the beautiful songs of cicadas. The carved wings are my little tribute to those wondrous musicians.

[2]`http://www.arduino.cc/`

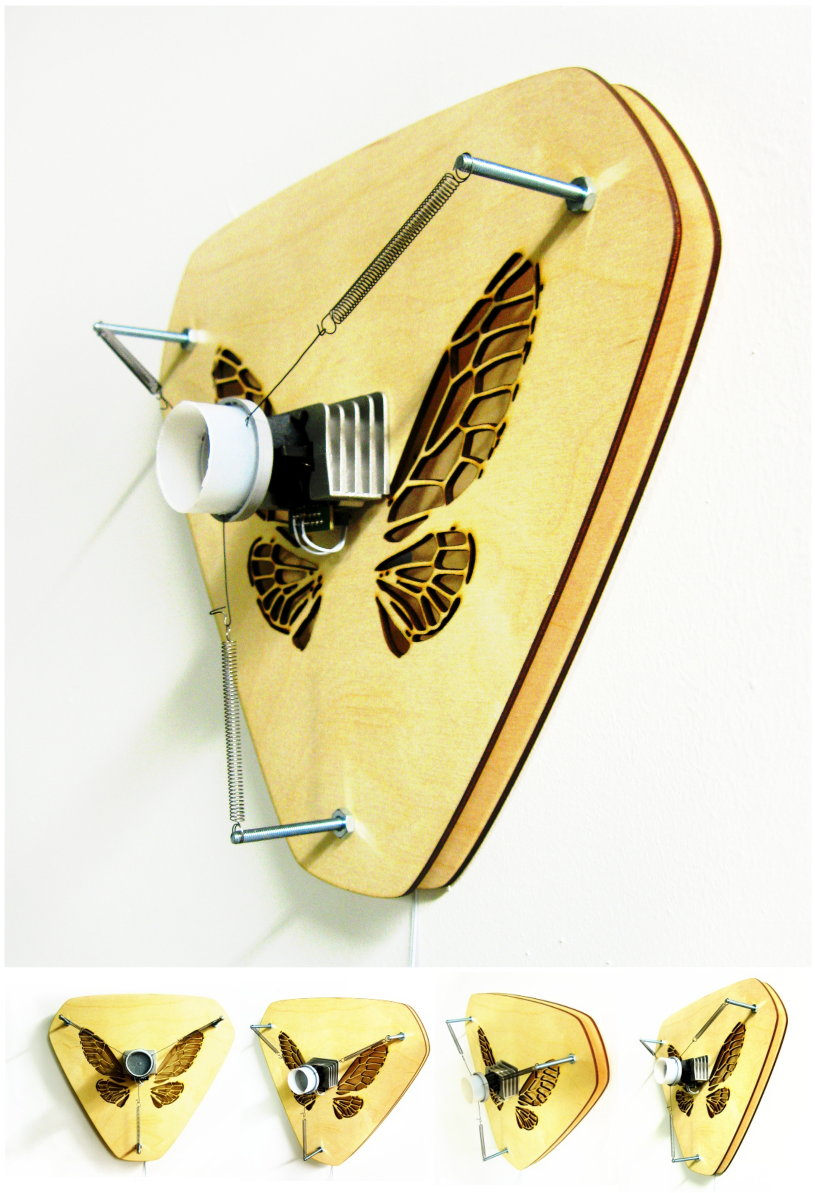


Figure B-1: Custom built 1-bit instrument. The instrument comprises a wooden base, a dot matrix print head, a plastic cap, wire and springs.

of the driver is an array of eight TIP120 transistors, each of which serves as a gate; when the Arduino sends a 5 volt impulse to a transistor, it in turn sends a 35 volt impulse to the corresponding print head solenoid.

The print head driver also comprises a 74HC595N shift register. This is essentially a bit array with sequential input and parallel output; the array is filled one bit at a time, and outputs all its bits simultaneously. Its purpose is to synchronize the multiple print heads connected to the board. The board is also designed so that multiple copies of it can be connected in series, allowing one to control more than eight solenoids simultaneously.

Figure B-2 shows the schematic of the dot matrix print head driver, and Figure B-3 shows the PCB design. Finally, Figures B-4 and B-5 show the final construction of the print head driver board and the Arduino micro-controller connected to it.

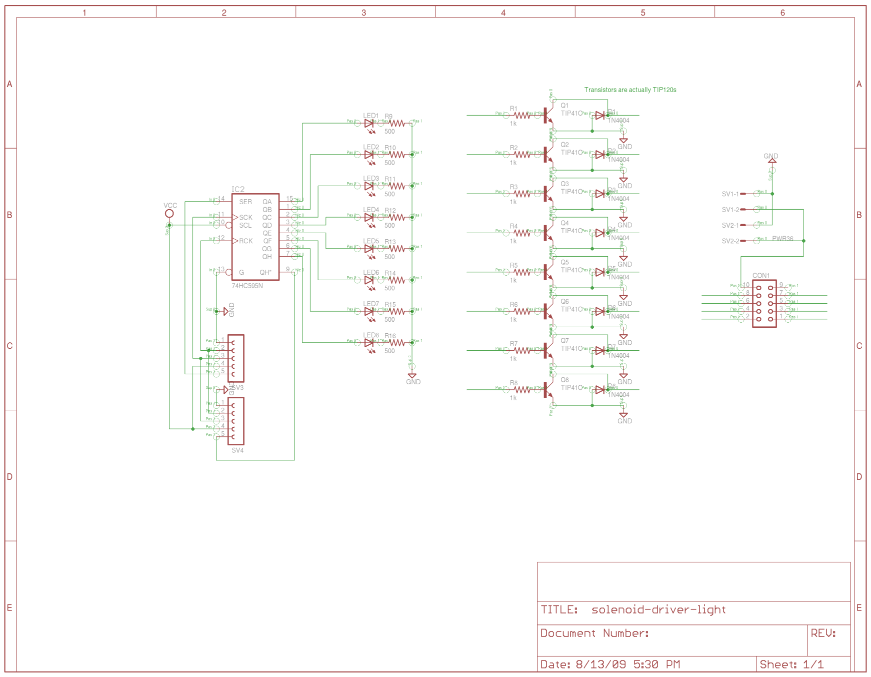


Figure B-2: Solenoid driver borad schematics. A 74HC595N shift register distributes 5 volt impulses to eight TIP120 transistors.

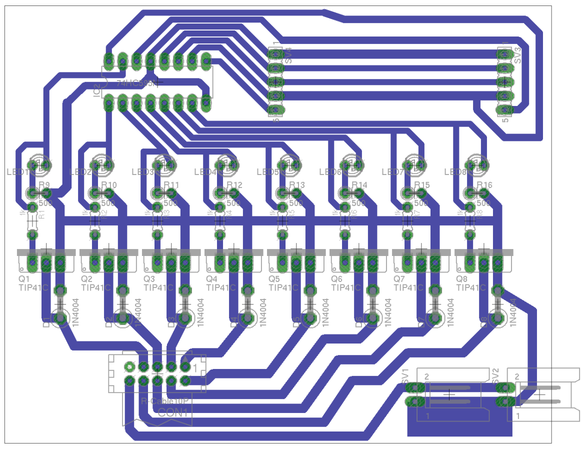


Figure B-3: Solenoid driver borad traces.

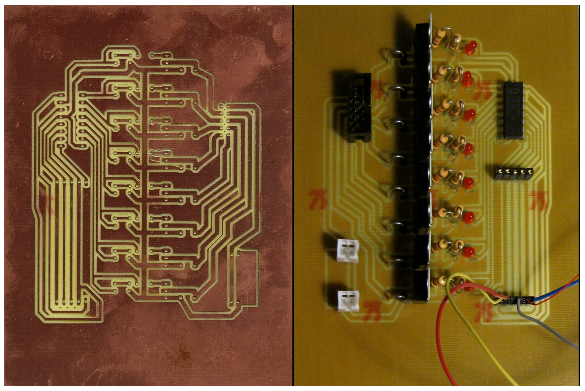


Figure B-4: Solenoid driver board construction. The pre-soldered board is shown from the bottom (left), and from the top with components added (right).

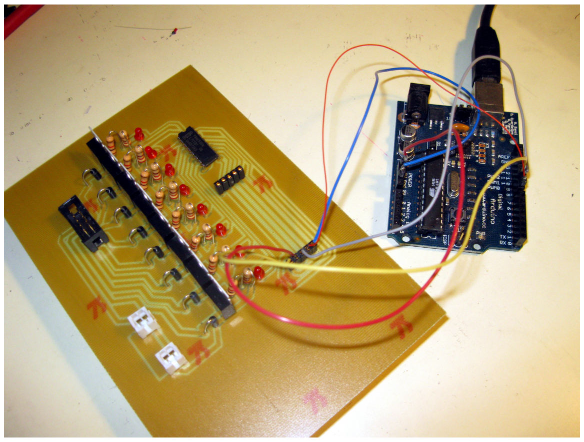


Figure B-5: Solenoid driver borad connected to the Arduino board.

Bibliography

[1] V. Adán. Xenakis' *Psappha*, 2009. Unpublished.

[2] G. Boole. *An Investigation of the Laws of Thought.* Project Gutenberg, 2005. First published in 1854.

[3] J. P. Crutchfield. The calculi of emergence: Computation, dynamics, and induction. *Physica D*, 75:11–54, 1994.

[4] J. P. Crutchfield and D. P. Feldman. Regularities unseen, randomness observed: Levels of entropy convergence, 2001. http://arxiv.org/abs/cond-mat/0102181v1.

[5] J. P. Crutchield and K. Young. Inferring statistical complexity. *Physical review letters*, 63(2):105–108, 1989.

[6] J. Estrada and V. Adán. La transformación continua de la forma de onda por medio del potencial combinatorio de sus intervalos de tiempo. Technical report, Universidad Nacional Autónoma de México, 2002. Presented at the International Simposium of Musical Acoustics, Mexico City.

[7] D. Feldman. A brief introduction to: Information theory, excess entropy and computational mechanics. Technical report, Department of Physics, University of California, Berkeley, 2002.

[8] H. G. Flegg. *Boolean Algebra and its Application.* Blackie & Son, Limited, 1964.

[9] E. Jones. Residue-class sets in the music of Iannis Xenakis: An analytical algorithm and a general intervallic expression. *Perspectives of New Music*, 39(2):229–261, 2001.

[10] A. Kraskov, H. Stogbauer, R. G. Andrzejak, and P. Grassberger. Hierarchical clustering based on mutual information, 2003. http://arxiv.org/abs/q-bio/0311039.

[11] W. A. Sethares. *Rhythm and Transforms*. Springer-Verlag, 2007.

[12] C. R. Shalizi. *Causal Architecture, Complexity and Self-organization in Time Series and Cellular Automata*. PhD thesis, University of Wisconsin at Madison, 2001.

[13] C. R. Shalizi. Blind construction of optimal nonlinear recursive predictors for discrete sequences. In M. Chickering and J. Halpern, editors, *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence (UAI 2004)*, pages 504–511. AUAI Press, 2004.

[14] B. Sklar. *Digital Communications*. Prentice Hall, 2001.

[15] S. W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Techinical Publishing, 1997.

[16] I. Xenakis. Towards a metamusic. *Tempo*, 93:2–19, 1970.

[17] I. Xenakis. Sieves. *Perspectives of New Music*, 28(1):58–78, 1990.