MUSICA THEORICA 2021, V. 6.1
SCIENTIFIC ARTICLE
DOI: 10.52930/mt.v6i1.192
Data do recebimento: 18/07/2021
Data da aprovação final: 18/09/2021

Codings for rhythm generation: a proposal and comparative study

Codificações para geração de ritmos: uma proposta e um estudo comparativo

Adolfo Maia Jr.
University of Campinas
Igor Leão Maia
Federal University of Minas Gerais

Abstract: In this work, we present a brief review of strategies to code rhythms and point to their possibilities and limitations in a unified way. We start by giving an overview of the representation (coding) of rhythms and their possible uses. Then we present different methods to analyse and generate rhythm patterns, which can be easily read by humans, through a simple algorithm. We also aim to provide a general evaluation of their pros and cons regarding their use in composition and analysis. In a more abstract approach, we define Rhythm Spaces as sets of strings of symbols endowed with suitable operations and algorithms that can be applied to generate new and complex rhythm patterns. Our approach can be useful in order to provide suitable code/notation to be used in computer applications in rhythm analysis and composition.

Keywords: Rhythm Encodings. Music Analysis. String Representation. Computational Musicology.

1. Introduction

In their now classical work, *The Rhythmic Structure of Music,* Cooper and Meyer have stressed in its first phrase that "to study rhythm is to study all the music. Rhythm both organizes, and is itself organized by, all the elements which create and shape musical processes" (Cooper; Meyer 1960, p.1). It is difficult not agree with them since music is a time process of evolving sound structures, which are constructed mainly taking into account the pace of their time transformations. Varese's moto "Music is organized sound" (1996) implies, most importantly, the time organization as mentioned by Cooper and Meyer. In many cases, rhythm is the fundamental structure of a musical piece.

Music notation, in most cases, tries to record visually the time perception of these sound structures by using correspondent strings of symbols for each instrument which are graphically represented them in a two dimensional score. However, music notation, although highly developed nowadays, as well as further accessible through computer software, are not quite appropriate for formal and computer manipulations from the point of view of composition, as well as for quantitative analysis of large sections of symbolic music data (music notation and scores). There are, however, some quantitative music analysis and incipient formal manipulations included into commercial software although they are not tailored to deeper analysis or composition with mathematical or formal approach. This was one of the principal motivations for the creation of the area of Computational Musicology, that is, to develop algorithmic tools able to find patterns in large collections of symbolic music data (Cook 2004). Closely related are the efforts in the area of Digital Humanities which, among other goals, make use of large collections stored in data banks as, for example, historical documents, letter collections and other kinds of material, many of them in pdf and other formats, which can be freely accessed and analysed. Such a project, for the case of musicological studies must include full digitalization of works of music in order to encode them in MIDI, XML or, possibly, in other formats, which can be reproduced and formally analysed through computer programs designed for music analysis and musicology. Such computer programs, of course, must recognize and manipulate melodic, harmonic and rhythmic structures. This work goes in the direction to provide some suggestions for coding rhythms.

Although rhythm manipulations have been explored since the beginning of music making itself, we think that there is a lack of a significant number of rhythm representational models which can be algorithmically implemented in a computer language. Along the past centuries, Western Music notation became highly complex in order to encompass ancient as well modern and contemporary music (Gould 2011), which has also incorporated sophisticated graphical notations as shown, for example in Cage's book Notations (Cage 1969), among many others. With the development of areas such as digital technology, computer music and, more recently, computer musicology, it became necessary to think how to code music efficiently (Cook 2004), mostly in order to get quantitative and statistical data quickly as well as formal manipulations for music composition.

Now, rhythm is a time organized phenomenon or, more formally, a time ordered set of sound events. Of course, any representation of rhythm, since it abstracts from the sound source, is a reductionist approach, nevertheless, it provides valuable information which the contemporary technology allows us to get qualitative and quantitative digital data which are amenable for computer approach in analysis and composition. In line with this somewhat reductionist scheme, in this work we are interested in studying rhythms from a mathematical point of view, taking into account a translation, through suitable representation, from Western Music Notation to strings of symbols which, in its turn, can be coded and used through some algorithm for computer based musical analysis as well music composition. However, we don't go deeper into algorithms in this paper, being our main purpose here to find ideally useful, and also minimal, representation for given rhythm patterns. So, in this paper, a rhythm pattern can be thought as a rhythmic motive or even a rhythm phrase.

An early and interesting step in this direction was given by Hook (1998) which, in order to study rhythm patterns in Messiaen's *Turangalila* Symphony, develops a simple code and operations on rhythm patterns. Toussaint (2013) presents many different notations of rhythm patterns and make extensive use of his geometrical notation in his approaches to the study of rhythm. However, this representation focus on cyclical rhythmic structures leaving behind other types of rhythmic structures. Sethares also presents, in Section 2 of his book, several different rhythm notations (Sethares 2007). Nevertheless, since these authors are not primarily concerned with rhythm notation itself, they don't present any code representations for arbitrarily complex rhythm. However, an exception to this is comprehensive study *Computational Models of Rhythm and Meter* by Georg Boenn (Boenn 2018), in which he presents a coherent "shorthand notation for musical rhythms" (SNMR), based on an early work by percussionist Peter Giger on "*rhythmoglyphs*". In addition, he also presents his software *Chunking* for composition and analysis, mainly for rhythm patterns, although he doesn't provide complex examples of second order nested rhythm patterns as our example in Fig. 11 below.

Most probably there is no mathematical rhythm representation, and consequently, no algorithm, capable to code all possible human creation of rhythm patterns and vice-versa. However, it's possible to get mathematical rhythm representations which partially do the job for a restrict domain as, for example, most part of Western Common Notation. The representation itself should allow the user to know the limits of its usefulness. Sethares reinforces the limits of formal representations as "attempts to mimic":

> A computational approach to the study of rhythm builds a model or a computer program that attempts to mimic people's behavior in locating rhythms and periodicities (Sethares 2007).

In this paper we are more interested in presenting alternatives for rhythm representation from moderate to reasonably high level of complexity. This means it can include variations of rhythmic structures such as tuplets and different patterns of notes and rests. To do so, we start by presenting a review of some common rhythm representations, as they appear in works from the above mentioned authors among others. Since all those representations are strongly based on generation and analysis of strings of symbols, we show in section 3 a brief introduction of the theory of strings of symbols, also nicknamed "Stringology" (Crochemore; Rytter 2002; Crochemore *et al*. 2007). Following the short exposition of "Stringology", we present an algebra for binary coded strings and later for triadic representations which include more complex rhythm patterns.

As commonly understood musical rhythm is perceived as the set of time distances between consecutive elements of a discrete sequence of sounds, or, more technically, between sound attacks. However, since we are most interested in the translation from music notation to strings of symbols, we do not intend, in this paper, to delve in the matters of psychoacoustic definitions of rhythm. The above particular definition, although incomplete, is enough for our formal approach of rhythm representation as a string of symbols.

2. Coding rhythms on strings

Formally, we are interested to work in a defined *Rhythm Space*. The formal approach to do this is based on the following:

1. Define, or take, a finite set of basic rhythms, say **b**. This is quite arbitrary, of course, however our approach is also very general, so no restriction is necessary at this step.
2. Define a finite set of operations **O** on **b**. Also these operations are also arbitrary, but they must be consistent and unambiguous when applied on elements of **b**. These operations, in general, will create new rhythm patterns.
3. A Rhythm Space ***R*** based on **b** is the set of all possible rhythm patterns generated through the set of operations of item 2. We can write the Rhythm Space as a pair ***R*** **= (O, b)**. Of course, the bigger **b** and **O**, the bigger **R**.
4. Now we can extend the set of operations from **B** to the Rhythm Space ***R***, as well create new kinds of operations. This extended set of operations allows new modifications and combinations of rhythms.
5. Once obtained a Rhythm Space it is possible to use its subsets to generate rhythm patterns for composition, analysis, or even parsing a given rhythm pattern in terms of basic ones.

Observe that although **b** and **O** are finite sets, the Rhythm Space ***R*** is not, since the operations in **O** can be applied an arbitrary number of times on any element, or subset of elements, of **b.** The idea of start from a basic set of rhythms is to have control on the possible outputs, so we can logically reproduce or modify any result. Also it resembles the mathematical idea of a basis of a Vector Space, or even generators of mathematical structures like groups or algebras.

Now, in order to get the formal approach as described above, a mathematical framework on which we can work out our model of rhythm theory is necessary. As mentioned before, this framework is based on the representation/coding and formal operations of rhythm patterns by strings of symbols, that is, through *Stringology.*

2.1. Simple String Representations

Out of all possible rhythm representations, the simplest one and most commonly used is the so-called binary encoding. An example of this type of representation is one in which the value ″1″ represents an attack (or ***note*** *on* in MIDI protocol) and "0", represents ***rests***. In this section we denote as $\boldsymbol{S}(\boldsymbol{n})$ the set of all strings using a set of $\boldsymbol{n}$ symbols. In this way, for example, Reich's famous Clapping Music rhythm pattern, shown in Fig. 1, is coded as a binary string, taking an eighth note as rhythmic unit, as $s = [1\,1\,1\,0\,1\,1\,0\,1\,0\,1\,1\,0]$. So, it's an element of $\boldsymbol{S}(\mathbf{2})$.

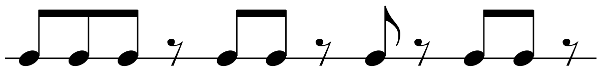


Figure 1: Reich's Clapping Music Rhythm.

However, this representation clearly has its limits. For instance, all notes and rests in Fig. 1, the Reich's rhythm pattern, have the constant duration of a eighth note. So, no symbol is necessary for duration. The eighth note duration works like a slot which can be full, with a note (symbol 1) or empty, (symbol 0). Now, consider the rhythm shown in Fig. 2. Strictly speaking it can't be represented only by using eighth notes, since note durations now are different one from another and the attacks are clearly non-periodic. The problem can be solved, for example, if we introduce an additional symbol for "time prolongation", meaning time linear augmentation or tie, which can differentiate between the basic rhythmic unit and other durations. However, in doing so, we must enlarge our alphabet from 2 to 3 symbols. For example, using the symbol "&" to denote the augmentation of basic rhythmic units, which can include ties, the rhythm pattern of Fig. 2, can be written as

$$s = [\,1\,\&\,1\,\&\,1\,0\,1\,\&\,1\,0\,1\,0\,1\,\&\,1\,0\,]$$

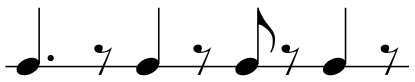


Figure 2: A rhythm pattern derived from Reich's string $s \in \boldsymbol{S}(\mathbf{3})$.

Observe in the above example that the code is the same for a punctuated quarter note or three tied eighth notes. In short, any string representation has some kind of limitation for more complex notations. For example, it is not difficult to write new rhythm patterns that also can't be written as strings using 3 symbols or any other number of symbols. It is worth to stress that our approach intends to get general representations which can be read by the user, since any representation can be ultimately reduced to binary code and then read by a computer program. However, this kind of binary representation would be very tricky to decode visually and, for sure, difficult to work with. So, our approach intends to be a useful tool in analysis or composition just by reading rhythm strings.

Therefore, complex rhythms require strings using a greater number of symbols. The above problem can be viewed in a more formal approach according to Shannon's Theory of Information which proves that new information, that is, a new symbol is needed in order to preclude ambiguity such that one shown above, in Fig. 2. In that case we offered obvious solutions, but they aren't the only ones. In fact, the new information, or symbol, needed to code them uniquely can be, for example, note duration, which we prefer than a "tie" symbol. Clearly, more complex rhythms may require a symbol for tied notes.

Now, non-accented simple rhythms like the above in Fig. 2 can be represented using only three symbols. For example, consider the representation which we take the number "1" for note attack, "-1" for rests, and the number of "0" (zeros) for note or rest duration. Accordingly, taking the time unit as the eighth note, the rhythm pattern in Fig. 2 can be coded as $\boldsymbol{S}(\mathbf{3})$ string:

[1 0 0 0 -1 0 1 0 0 -1 0 1 0 -1 0 1 0 0 -1 0]

Observe that both representations of the rhythm pattern in Fig. 2 need three symbols. However, the interpretation is different for the symbols and both strings represents the rhythm pattern uniquely.

Now, observe that the above binary encoding doesn't take into account rhythm accentuation. Again, if new information is needed, such as for an accentuated rhythm pattern, this implies introducing a new symbol for accentuation. Particularly, one can suffice most musical needs in term of accentuation by thinking in binary code, that is, accented or no accented notes. This can be accomplished changing the encoding of note attack as, for example: keep the symbol "1" for non-accentuated notes and "2" for accentuated ones. For example, consider the rhythm pattern of Fig. 3.

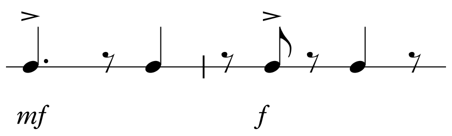


Figure 3: Accentuated rhythm pattern.

It can properly be coded using 4 symbols as

[2 0 0 0 -1 0 1 0 0 -1 0 2 0 -1 0 1 0 0 -1 0]

As another example, a simple rhythm pattern like that one shown in Fig. 4 can be coded with just 4 symbols, where "2" means an accented note, without taking into account any of the information on different hierarchies of accentuation or dynamics. This is the minimal alphabet for our approach of accented rhythms.

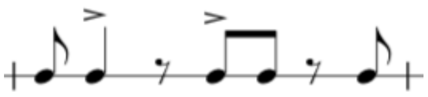


Figure 4: A simple rhythm pattern with code string x = [1 0 2 0 0 -1 0 2 0 1 0 -1 0 1 0].

Observe that $x \in \boldsymbol{S}(\mathbf{4})$. In the above examples we chose an eighth note duration as time unit. Of course, the smaller the time unit the greater the quantity of the symbol **0** needed to represent larger durations.

Now, another question is how to include dynamics? Firstly, observe that in order to represent properly dynamics in rhythm patterns it is necessary to include at least two other symbols: one indicating the group of notes to change dynamics and other ones denoting the dynamics levels. Therefore, in the case of N levels of dynamics, N+1 symbols are necessary. For example, let's consider just one level of dynamics, say, a forte, denoted by f. We can use, for example, a symbol like a bar | in order to delimit the group of notes whose dynamics we want to change. Now we have a set of six symbols at disposal, namely, $S = \{-1, 0, 1, 2, f, |\}$. The only rule for grouping is that the first symbol inside the bars is the dynamics symbol. Consider, for example the rhythm pattern in Fig. 5.

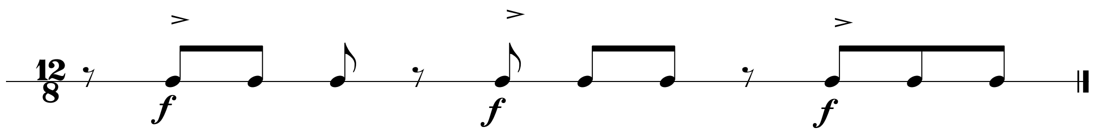


Figure 5: An example of pattern including dynamics.

The code for this pattern is given by a $\boldsymbol{S}(\mathbf{6})$ string:

$$\boldsymbol{s} = [\mathbf{-1\ 0\ |\ \mathit{f}\ 2\ 0\ |\ 1\ 0\ 1\ 0\ \ -1\ 0\ |\ \mathit{f}\ 2\ 0\ |\ 1\ 0\ 1\ 0\ \ -1\ 0\ |\ \mathit{f}\ 2\ 0\ |\ 1\ 0\ 1\ 0}\ ]$$

The above approach can be generalized without conceptual difficulty using a finite set of symbols, that is, an *alphabet*, depending on the number of musical parameters and their different levels. The greater the alphabet and/or the allowed length of the strings the greater the possibility for the representation, or encoding, of complex rhythms. Or in other way around, the greater complexity of rhythm patterns the bigger the set of symbols needed to represent them as strings of symbols.

It is important to observe that not all strings formed with a given alphabet represents a rhythm pattern. In fact, the valid ones form a proper subset of the entire set of possibilities. For example, Figs. 4 and 5 show simple accented rhythm patterns with their associate code, but the string y = [0 2 0 0 -1 0] doesn't represent any rhythm string since it begins with a duration symbol and not with an event symbol such as note or rest. Another example, the string $w = [\ 1\ 0\ 1\ 1\ 0 - 1\ 0]$ also doesn't represent any rhythm patterns since it has two consecutives attacks without their following zeros in order to inform duration. Now, the string $z\ =\ [2\ 0\ 1\ 0\ \ -1\ 0\ 1\ 0]$ does represent the rhythm pattern formed with the last four notes/rests in Fig. 4, since all the rules are fulfilled. It's easy to see that all rhythm strings must end with 0, since any event must have a duration indication. In addition, of course, there is no accent for rests. In short, not every element of **S(n)** represents a rhythm string. In fact, rhythm strings, written with a set of n symbols, is a proper subset of **S(n)**.

As another example of alternative notation, consider the rhythm pattern of Fig. 2 (without accent). It can be coded as a $\boldsymbol{S}(\mathbf{3})$ string

$$\mathbf{|\ 1\ 0\ 0\ 0\ |\ 0\ 0\ |\ 1\ 0\ 0\ |\ 0\ 0\ |\ 1\ 0\ |\ 0\ 0\ |\ 1\ 0\ 0\ |\ 0\ 0\ |}$$

Here notes are separated by the small bar, where the first symbol inside a pair of bars means the type of event, that is, 1 = note, 0 = rest. The first symbol, either 1 or 0, gives a note or a rest while the number of other 0 symbols means the duration in terms of the time unit. Observe that the symbol 1 must appears always at first position and it is always interpreted as a note, but the interpretation of the symbol 0 depends on its position (at first position or out of there). Observe that again we needed 3 symbols, except that the meaning of them were changed, but the size of the alphabet is the same. In terms of the Theory of Information there is no difference between them. In Sections 3 and 4 we develop further these concepts with several examples.

2.2. Rhythm as Arrays

It is clear now that the greater the number of music parameters the greater the number of symbols necessary to represent them in a single string. A way to circumvent that increasing of the number of symbols, in our rhythm alphabet is to encode parameters such as accents, or dynamics, as additional binary strings or, equivalently, defining a matrix representation with two, or more rows where the first row is the basic representation of the rhythm pattern, that is without accents or dynamics or other parameter. These extra information is represented as added strings below the first one. Consider, for example the rhythm pattern in Fig. 5, which includes accentuation and dynamics. The first row is the usual binary representation of note/rest with the symbols 0 and 1. In the second row, the accent is coded as symbol “1” below the note attack, the other ones take the value “0”. The third row shows the representation of dynamics as “1” for the note with changed dynamics and “0” otherwise. In this example they coincide and we have just one level of dynamics (one symbol). It can be coded as the following matrix

$$\begin{bmatrix} -1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 6: Matrix Representation of Rhythm Pattern in Fig. 5. The first row is the non-accented rhythm, the second one gives the place of accents and the third one, the place of dynamics, here, just one, the forte accent.

Fig. Now, observe the above matrix represents only one bar. If we want more bars, each of them must be represented by three rows. For example, for the pattern in Fig. 5, if a second bar is the repetition of the first one, the matrix representation of the two bars pattern reads

$$\begin{bmatrix}
-1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

Figure 7: A representation of two bars of rhythm pattern in Fig. 5.

Observe also that time signature is not informed. This kind of information must be known a priori in this representation. In terms of the Theory of Information, we are using less symbols, only three, but the size of the concatenated string *[Row1, Row 2, …, Row 6]* representing the two dimensional matrix is six times bigger. In fact, we are using "four" symbols, the fourth would be, for example, a comma to differentiate one row from another, but this fact is "hidden" in the two dimensional representation. So, in certain sense, we don't get any economy in doing so. In fact, the best representation depends very much on the purpose of the user and the characteristic of applications. An advantage of matrix representation is that they are valuable for operations on the rows which are associated to independent musical parameters.

If we consider the use of this representation in composition, for instance, it can generate some possibilities for creative work. It can provide, for example, some hints for the composer on possible solutions to the relation rhythm/pitches defining also a correspondent matrix representation of set pitches sequences. However, these possibilities are not without its own problems. Limitations appear, for example, in difficulties to represent more complex rhythm structures which can require very large matrices. This problem is partially alleviated due to the current processing power of computers, nevertheless code readability is lost. In addition, formal methods, in general, don't have direct implications on aesthetics or style. As the masters of experimental and formalized music of the past have showed in several of their works, formal methods do not entail abdication of the composer's musical ideas, aesthetics and imagination.

3. A Short Introduction to Stringology

In this section we present a short introduction of a general theory of strings, their properties and operations. Our approach is based on that of Crochemore and Rytter (2002) and Crochemore *et al*. (2007). In the next two sections we apply it in a particular type of "stringology" of rhythms in which we include rhythmic representation for tuplets and other complex rhythms. The concepts defined in this section will be used in the next section when we will apply them to rhythm strings. The definitions are the following:

1. An *alphabet* A is a finite set of symbols (or letters).
2. A *string*, or *word*, is any finite sequence, constructed by juxtaposition, of symbols of the alphabet A. We denote an arbitrary string x with n elements as $x = x[1]\,x[2]\,x[3]\dots x[n]$. The i-th symbol (or element) of a string x is denoted by $x[i]$.
3. The set of all strings on the alphabet A, that is the total space of all possible strings, is denoted by A^*. The alphabet used to write a string x is denoted as $alph(x)$.
4. The *length* of a string x, denoted by $|x|$, is the number of its symbols (also named elements), including their repetition.
5. The *empty string* (no symbols) is denoted by ε.
6. x is a *substring* of y if x can be obtained from y by removing zero or more symbols (not necessarily adjacent) from it. This implies that if x is a substring of y then x can be written as $x = y[i_1]y[i_2]\dots y[i_m]$ for an increasing sequence of indices $i_1, i_2, \dots, i_m$.
7. As a special case of string we define a *factor* of x, denoted by $x[i \dots j]$, the substring $x[i]x[i+1]\dots x[j]$ extracted from x. If, $i > j$, the substring $x[i..j] = \varepsilon$, by convention. Equivalently, x is a factor of y if there exist two strings u and v such that $y = uxv$. In the case that $u = \varepsilon$, that is, $y = xv$, we say x is a *prefix* of y. If $v = \varepsilon$, that is $y = ux$, we say that x is a *suffix* of y.
8. Identity: $x = y$ if and only if $|x| = |y|$ and $x[i] = y[i]$ for $i = 1, 2, .., |x|$. A substring, or factor x of a string y is named as *proper* if $x \neq y$.

3.1. Operations on Strings

Let be an alphabet A and denote $S = A^*$, the set of all strings written with the alphabet A. An operation on the String Space S is just a function $f: S^m \to S^n$, where S^m and S^n are Cartesian products of String Space S. So the function f can have one or more parameters and can take a vector of strings as values. In the case it has just one parameter and one value, that is, $f: S \to S$, it is called an *Operator* on Rhythm Space S. Clearly, any function must be implemented with a suitable encoding, algorithm and code. Although there exists a great number of operations, some are very natural and useful, which we define below.

1. *Product (or Concatenation)*: Given two strings in S, $x = x[1]\,x[2]...x[n]$ and $y = y[1]y[2]...y[m]$, the product xy is just the concatenation of the strings, that is:
$$xy = x[1]\,x[2]...x[n]\,y[1]\,y[2]\,...y[m]$$

This function is, of course, of type $f: S{\times}S \rightarrow S$. Observe that the product is not a commutative operation, $xy \neq yx$, unless x = y. Nevertheless, it is associative. Also, the neutral element ε is just the empty string, since $x\varepsilon = \varepsilon x = x$.

2. *Power:* $x^0 = \varepsilon$, and for $n \geq 1$, we define recursively: $x^k = x^{k-1}x$ for k = 1, 2,..., n. This function is, clearly, an operator $f: S \rightarrow S$.
3. Reverse: If $x = x[1]\,x[2]...x[n]$ its reverse is defined as $rev(x) = x[n]\,x[n-1]...x[1]$. It is also an operator.

3.2. Stringology of Rhythms

In this section we show how to apply string operation and algorithms to the case of rhythms represented as strings with the alphabet $A = \{-1, 0, 1, 2\}$ as defined in Subsection 2.1. In order to do this, we must define what we mean by a *Rhythm String* based on the above alphabet A. Of course this can be generalized for an arbitrary number of symbols, with the suitable interpretations for rhythm patterns. See section 4, below, for an extension and its application on rhythm representation.

Definition: A *Rhythm String* is a string based on an alphabet A, satisfying the following rules:

1. The first element of a string is a non-zero value.
2. After a non-zero integer only zeros can occur and their number indicates the duration of the note or the rest.

Observe that all rhythm strings must terminate with 0 value, which indicates the last unit time of the last note of the pattern. Accordingly, the rules above also must be satisfied by *rhythm factors* of rhythm strings. Following the definitions in subsection 3.1 we denote as R the set of all *Rhythm Strings* based on the alphabet A, which we name ***Rhythm Space***. Clearly $R \subsetneq S$, since there exist strings in S which do not belong to R as, for example, any string starting with the symbol 0.

As a simple example, consider the rhythm and its code string as shown in Fig. 4. From the representation $x = [1\,0\,2\,0\,0\ -1\,0\,2\,0\,1\,0\ -1\,0\,1\,0]$ we can extract the factor

$$z = [2\,0\,1\,0\ -1\,0\,1\,0]$$

which corresponds to the last 4 figures of the score. Clearly z is a suffix of $y = [1\,0\,2\,0\,0\ -1\,0]$ in x and y is a prefix of z in x and, of course, we have the product $x = yz$.

Now, observe that, in general, the usual reverse of rhythm strings, as defined in Section 3.1, it is not a rhythm string, since, by rule 2 above, the last element of a rhythm string must be 0 and then its reverse must start with 0 which contradicts the rule 1. For example, $rev(z) = [0\,1\,0 - 1\,0\,1\,0\,2]$ doesn't obey the rule 1, and then it isn't a rhythm string.

Clearly we need to define an inversion of different kind. The new reverse is, in fact, the usual retrograde of a rhythm pattern encoded as a rhythm string, namely, given a rhythm string $x = [\,x[1]\,x[2]\dots x[n]]$, we define its retrograde, denoted as $ret(x)$, as the rhythm string constructed with the following simple algorithm:

1. Read the string x from right until find a nonzero value. Copy this block of values from left to right and paste it as the first factor of $ret(x)$.
2. Repeat the same operation for the next blocks always reading them firstly from the right, copying from the left, and paste them successively from left to right in the new string $ret(x)$.

For example, the retrograde of the rhythm string in Fig. 4 is

$$ret(x) = [1\,0\ -1\,0\,1\,0\,2\,0\ -1\,0\,2\,0\,0\,1\,0].$$

Observe that decoding this string in score notation we get exactly the retrograde rhythm of Fig. 4 as shown in Fig. 8.

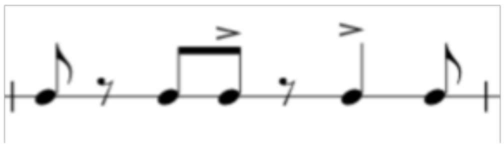


Figure 8: Retrograde rhythm pattern of Fig. 4.

Of course, the above algorithm can be implemented in any computer language as C++, Python, MATLAB among others, and the rhythm representation decoded in Common Notation by a program suitably designed to this task.

Below we present a pseudo-code which mimics a human by reading a code and translating to music notation. In fact, the source string and the output are just two ways to write the same rhythm pattern.

```
input string s = [s[1], s[2],…, s[n]]
durations = [ 0 0 … 0 0] // initial vector (size of s) with zeros only
for i= 1,2,…,n
if s[i] = 1
     duration[i] = number of following zeros until find next nonzero number (positive for notes)
if s[i] = -1
     duration[i] = - number of following zeros until find next nonzero number (negative for rests)
if s[i] = 0
    durations[i] = 0
notes = vector extracted from durations with only the nonzero values
```

For example, the above algorithm makes the following transformations on a rhythm pattern representation s, in $R(3)$

$$s = [1\,0\,1\,0\,0\,1\,0\,-1\,0\;\;0\,1\,0\,1\,0\,1\,0] \Rightarrow s' = [1\,0\,2\,0\,0\,1\,0\,-2\,0\,0\,1\,0\,1\,0\,1\,0] \Rightarrow$$
$$notes = [1\;\;2\;\;1\;\;-2\;\;1\;\;1\,1]$$

which translated to music notation reads (taking a quarter note as time unit)

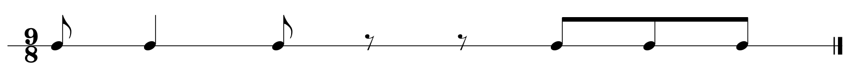


Figure 9: Rhythm pattern decoded from a string s in $R(3)$.

For simple, non-nested, rhythm patterns, just the sequence of durations of notes and rests is enough to easily write down the rhythm pattern or the sequence can be transformed into a MIDI notation and read through a music notation software.

In addition, it is possible to define algorithms to execute any formal transformation on strings and consequently endowing correspondent transformations on rhythm patterns. For example, below we show an algorithm, that is, a pseudocode, for the operation of retrograde on strings with the alphabet $A = \{1,0,-1\}$ as defined above.

```
input s = [s[1], s[2],…, s[n]] // enter the rhythm vector
for i= 1,2,…,n
r[i] = s[n-i] // inversion of the of the rhythm vector
r = [r[1], r[2],…, s[n-1], r[n]] // write the inverted vector
// Obs: r starts with a zero value (the number of zeros measures note duration). So it is not yet a rhythm vector.
Now, read vector r, until find the first non-zero value.
Invert this block $b_1$ and store it in a vector output.
Do the same process for other nonzero values of vector r. The result is
output = $[b_1\ b_2\ \dots\ b_k]$   // k blocks.
// Obs: Each block start with value 1 following with the number of zeros correspondent to the note duration
End of pseudocode
```

Of, course, the greater the complexity of the representation and operations the greater the complexity and size of the algorithm and its code. The advantage of a code which can be promptly read by a human, as the examples above, is the possibility of an easier search for patterns in music analysis and also the possibility to apply many transformations on patterns in the case of composition.

So, in the next section we show how to introduce representations for more complex rhythm patterns.

4. An Extended Code for Complex Rhythm Strings

Complex rhythms can have disparate combinations and variations as, for example, duration figures side by side like a whole and thirty-seconds notes. If we use the minimal figures as time units, the rhythm strings turn out to be very long, hampering readability. In order to overcome this problem, we use nested brackets as a notational solution, much the same the nesting symbol for tuplets in common music notation. This solve partially the problem since the grouped figures in tuplets can now have the same notation for duration as those outside the nest. In this way we can avoid using the very minimal duration figure of the score.

The idea in this section is to present a code that could be used for formal and computer applications as well as could be perused by a human. These two goals are in some sense opposed and resembles a little what happens in design of higher level computer languages. The closer to machine language the lesser the readability. We show our solution below.

4.1 Definition of the Extended Code

We present here an extension of a representation of rhythm patterns shown in Subsection 2.1. In order to do so, we define the alphabet and rules for construct more complex rhythm strings. For each *rhythm pattern* in a score we associate a *rhythm string* based on the alphabet and rules below. For our representation, the rules for rhythm strings are the following:

1. As in section 2.1, non-accentuated rhythm strings are coded with a pretty small alphabet, namely, A = {|, [,] , -1, 0, 1 }. If we include N levels of accent the alphabet is given by A = {|, [,], -1, 0, 1, 2, …, N}. | is the bar delimiter. The square brackets, **[** and **]** are used as delimiters of rhythm strings and their substrings (or nested substrings) related to rhythm patterns comprising a bar. They are useful to encode more complex rhythm structures such as tuplets or nested tuplets, in a bar.

2. If more information is needed as, for example, time signature and time unit we must add a *prefix* to the rhythm string with other symbols (mostly, numbers) in order to inform those parameters. In this case, the prefix to the rhythm string is coded as $[\,p]\,[\,q\,]\,[\,u\,]$, where p/q is the *time signature* and $\boldsymbol{u}$ is the *time unit* and p, q, u are positive integers. The time unit is, in general, the minimal non-nested duration used in the rhythm pattern, in order durations always have an integer number of zeros.
3. The alphabet needed to encompass all possibilities of prefix strings is far bigger than A, since, for example, in Common Music Notation we have 7 possible durations and many possibilities for time signatures. However, it is possible to restrict the alphabet to a suitable set of them depending on the rhythm content of the score, or the composer intentions.
4. The symbol $*$ denotes a tie joining two notes, be them within a beat, between beats, a bar or between two bars.
5. Typically, a rhythm string representing a rhythm pattern in a bar reads:

   $[\,\boldsymbol{p}\,]\,[\,\boldsymbol{q}\,]\,[\boldsymbol{u}]\,|\,[[\,\boldsymbol{structure}\ \ [\boldsymbol{substructure}][\boldsymbol{substructure}]]\ \ \ldots.\boldsymbol{structure}\ \boldsymbol{N}]\,|$

   where the first two bracket numbers $[p][q]$ of the prefix indicate the time signature as, for example, [7][16] means 7/16 signature. In principle, the time unit $\boldsymbol{u}$ is quite arbitrary, independent of the time signature**.** However, its common use is restricted to the usual 7 durations, that is, $\boldsymbol{u} = 1$ for Sixty-fourth Note, $\boldsymbol{u} = 2$ for Thirty Second Note, $\boldsymbol{u} = 3$ for Sixteenth Note, $\boldsymbol{u} = 4$ for Eighth Note, $\boldsymbol{u} = 5$ for Quarter Note, $\boldsymbol{u} = 6$ for Half Note, $\boldsymbol{u} = 7$ for Whole Note.
6. Nested brackets indicate hierarchy of grouping strings and substrings inside the set of beats indicated by the zeros of the rhythm string. Isolated notes inside a string or sub- string has none bracket.
7. In order to code also nested tuplets, notes inside second level nested substrings have half of value of the higher level but keeping the same coding as time units, that is [1 0] for a note, and [-1 0] for rests. See example 2 below.

Some examples can clarify our definitions.

Example 1: Consider the rhythm pattern shown in Fig. 10. The time unit is the eighth note since it is the minimal duration out of any nesting.

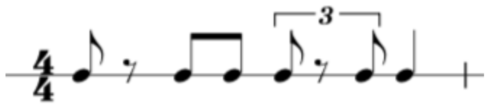


Figure 10: Rhythm Pattern with a triplet.

The associate rhythm string reads:

| [4][4][4] [1 0 -1 0 1 0 1 0 [1 0 -1 0 1 0] 1 0] |

Observe that we can't choose the time unit longer than the eighth note (the time unit, in this example), since in our representation the duration is given by an integer number of zeros.

Example 2: Fig. 11 shows a more complex, not accented, rhythm pattern using nested tuplets and ties and also with a double nesting.

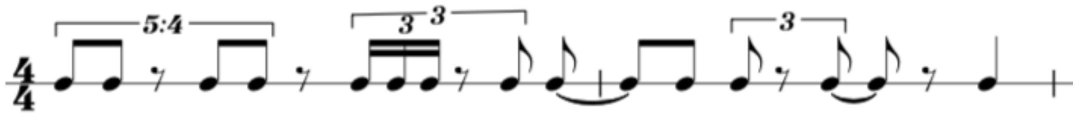


Figure 11: Rhythm Pattern with tied notes and single and nested triplets and a 5-tuplet.

From the above rules, its code reads:

[4][4][4] | [1 0 1 0 -1 0 1 0 1 0] -1 0 [[1 0 1 0 1 0] -1 0 1 0] 1 0 | * |1 0 1 0 [1 0 -1 0 1 0] * 1 0 -1 0 1 0 0 |

Observe that the substring **...[[1 0 1 0 1 0]...** is a second level nested rhythm pattern, so the values of its notes are half of the chosen time unit, that is, in this example they are sixteenth notes as expected in common notation. Observe also the position of symbol ∗ in the rhythm string representing ties joining notes within and between bars. Although it is a logical step, we will not take further to third level nesting which presents additional difficulty in order to avoid ambiguity for these types of grouping of notes.

4.2. Elementary Operations on Rhythm Strings based on the Extended Code

Given a rhythm pattern (or motive), there are many formal operations which can be applied to a single string (unary operation), between two strings (binary operation) and more generally N-ary operation on a set with N strings. It worth to mention that a formal operation can lead to a correct but not elegant representation. For example, the operation Note-Rest Inversion, defined below, can produce two or more consecutive rests which, in general can be rewritten in a more suitable form.

Below we show some simple operators which can be applied to rhythm patterns and side by side can be defined on the corresponding rhythm strings. In order to show examples, consider the rhythm pattern in Fig. 12, extracted from the first bar of Fig. 11.

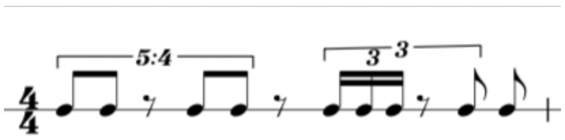


Figure 12: Rhythm pattern extract from first bar of Fig. 11.

with rhythm string

x = [4][4][4] | [1 0 1 0 -1 0 1 0 1 0] -1 0 [[1 0 1 0 1 0] -1 0 1 0] 1 0 |

1. ***Augmentation:*** just change the time unit keeping all other symbols unchanged. So, for the rhythm pattern above we have

   ***aug*(x) = [4][4][5] | [1 0 1 0 -1 0 1 0 1 0] -1 0 [[1 0 1 0 1 0] -1 0 1 0] 1 0 |**

which is decoded as

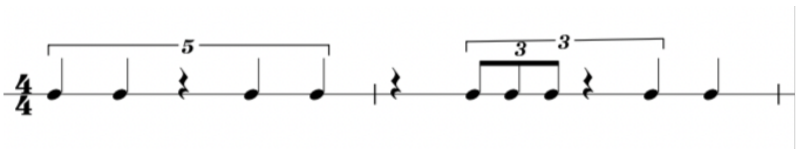


Figure 13: Augmentation of rhythm pattern in Fig. 12.

An alternative representation for the same rhythm pattern of Fig. 13 is just keep the prefix, including time unit, and add the desire number of zeros for each non-zero number. For example, adding one more zero we get

aug(x)=|[4][4][4] | [1 0 0 1 0 0 -1 0 0 1 0 0 1 0 0] | -1 0 0 [[1 0 0 1 0 0 1 0 0] -1 0 0 1 0 0] 1 0 0 |.

Note that, in the case of augmentation, the time unit gets increase by 1, that is, it's a quarter note, in the prefix and the rhythm pattern now comprises two bars.

2. ***Diminution:*** since the original rhythm pattern has an eighth note as time unit, the new one has a sixteenth note as time unit, so the modification is just the following pattern

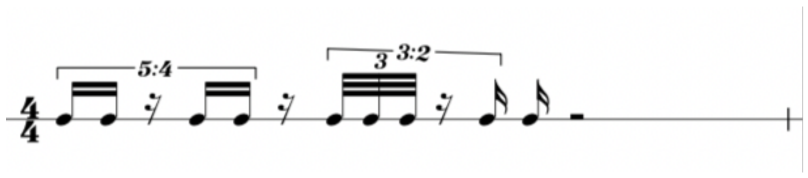


Figure 14: Diminution of rhythm in Fig. 12.

and its representation code reads

dim(x) = [4][4][3] | [1 0 10 -1 0 1 0 1 0] -1 0 [[1 0 1 0 1 0] -1 0 1 0] 1 0 -1 0 0 0 0 0 0 0 0] |

Observe that, differently from the case of augmentation, we can't have an alternative representation keeping the unit time unchanged, since we got fractional numbers to represent sixteenth notes.

3. ***Retrograde*:** by the above definition of retrograde we have, from Fig. 12, the following rhythm pattern

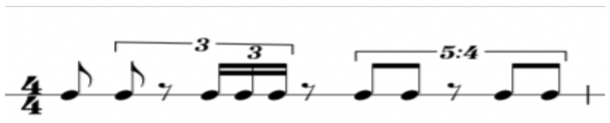


Figure 15: Retrograde of rhythm in Fig. 12.

and its code reads

ret(x) = [4][4][4] | 1 0 [1 0 -1 0 [1 0 1 0 1 0]] -1 0 [1 0 1 0 -1 0 1 0 1 0] |

Observe that the prefix is exactly the same.

4. ***Product:*** Given two rhythm-strings *x* and *y* we can use the string product (non commutative), defined above as juxtaposition, in order to get large rhythm patterns, that is *xy* or *yx*. Observe that strings *x* and *y* can have different prefixes, as change of metrics from a bar to the next one. So they need to be written before each string. For example, consider the rhythm pattern

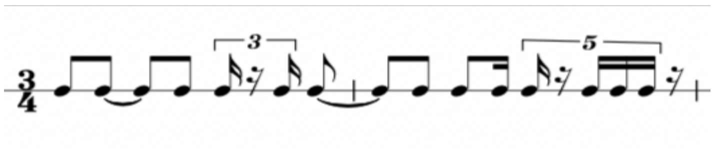


Figure 16: Rhythm pattern as the first factor of the example of Product operation.

It has a sixteenth note as time unit. Let's denote its code as **x,** which reads

**x = [3][4][3] | 1 0 0 1 0 0 * 1 0 0 1 0 0 [1 0 -1 0 1 0] 1 0 0 | *
| 1 0 0 1 0 0 1 0 0 1 0 [1 0 -1 0 1 0 1 0 1 0] 1 0 |**

Also consider the rhythm pattern of Fig. 15 and its code

y = [4][4][4] | 1 0 [1 0 -1 0 [1 0 1 0 1 0]] -1 0 [1 0 1 0 -1 0 1 0 1 0] |

The product **xy** reads, by juxtaposition,

xy = [3][4][3] | 1 0 0 1 0 0 * 1 0 0 1 0 0 [1 0 -1 0 1 0] 1 0 0 | * | 1 0 0 1 0 0 1 0 0 1 0 [1 0 -1 0 1 0 1 0 1 0] 1 0 | [4][4][4] | 1 0 [1 0 -1 0 [1 0 1 0 1 0]] -1 0 [1 0 1 0 -1 0 1 0 1 0] |

which is decoded as the score fragment

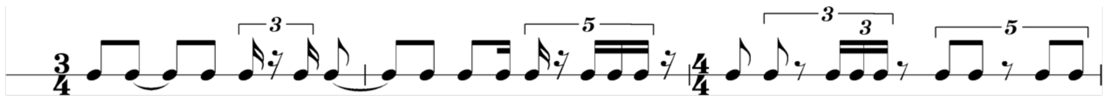


Figure 17: Fragment obtained as product (juxtaposition) of rhythm patterns of Figs. 16 and 15, respectively.

5. ***Fragmentation*** *(or* ***Truncation****):* extract any figure, or set of figures, from the rhythm pattern. Consider again the rhythm pattern in Fig. 16 and extract, for example, the rhythm block

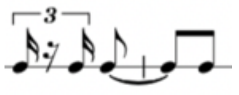


Figure 18: A fragment extracted from rhythm pattern in Fig. 16

The code for this fragment reads simply as

[3] | [1 0 -1 0 1 0] 1 0 0 * 1 0 0 1 0 0 |

Observe that only the time unit of the prefix must be informed, since the extracted excerpt from the score has no indication of time signature. Also, the vertical bars | | indicate the begin and the end of the excerpt not a measure as shown in the above figure.

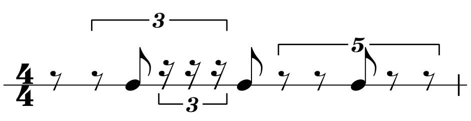
6. ***Note-Rest Inversion****:* exchange all notes to rests and vice-versa. This means exchanges 1 by 0 and vice versa. As example, take the rhythm pattern of Fig. 15.
   Its code reads

[4][4][4] | 1 0 [1 0 -1 0 [1 0 1 0 1 0]] -1 0 [1 0 1 0 -1 0 1 0 1 0] |

Its Note-Rest Inversion reads:

[4][4][4] | -1 0 [-1 0 1 0 [-1 0 -1 0 -1 0]] 1 0 [-1 0 -1 0 1 0 -1 0 -1 0] |

which is decoded as the rhythm pattern



As a general observation, note that in this representation for any rhythm code the number of left brackets **' ['** is equal to the right brackets ' **]** '.

Of course, this representation does not necessarily include all possible rhythms. It has its limitations, including the choice of the unit rhythmic figure.

5. Using the Extended Code for Analysis and Composition

In this section we just show a diagram indicating the use of rhythm codes in analysis and composition. The source, or coding, comes from a score in MIDI or XML format. While in MIDI-based representation the time is a continuous variable and note durations are controlled by Note-On and Note-Off specification, XML based representation has durations indicated closely to that one from the score. So, they are more suitable for our proposal of rhythms coding. The diagram below shows the process from the input score, or directly coding a rhythm pattern (AUTO), to the output score (see Fig. 17).

In the case of computer aided composition, a natural continuation of this work could be in the generation of new rhythm patterns using the concept of *Rhythm Algebras* defined on sets of strings and how to use them to generate complex rhythm patterns. These Algebraic structures enters in the Diagram as the FORMAL/MATH MODELS. A simple algebra can be generated, for example, as combinations of the operations defined in Section 4.2.

Another interesting concept which can be explored using the code is that one of path or orbit in a Rhythm Space. Given an initial rhythm pattern x_0 and an operator T, an orbit of size N is a sequence of points (rhythm patterns) in the Rhythm Space given by the recursive application of operator T, that is,

$$\begin{gathered} x_1 = T(x_0) \\ x_2 = T(x_1) \\ x_3 = T(x_2) \\ \dots \\ x_N = T(x_{N-1}) \end{gathered}$$

The orbit is closed (or cyclical) if there exist a $N > 0$ such that $x_N = x_0$. For example, the operator Retrograde generate always orbit of size 2, since, given any rhythm pattern x_0 we have $x_1 = R(x_0)$ and $x_2 = R(x_1) = R(R(x_0) = x_0$.

The definition above can be extended for a set of operators or/and a set of initial points.

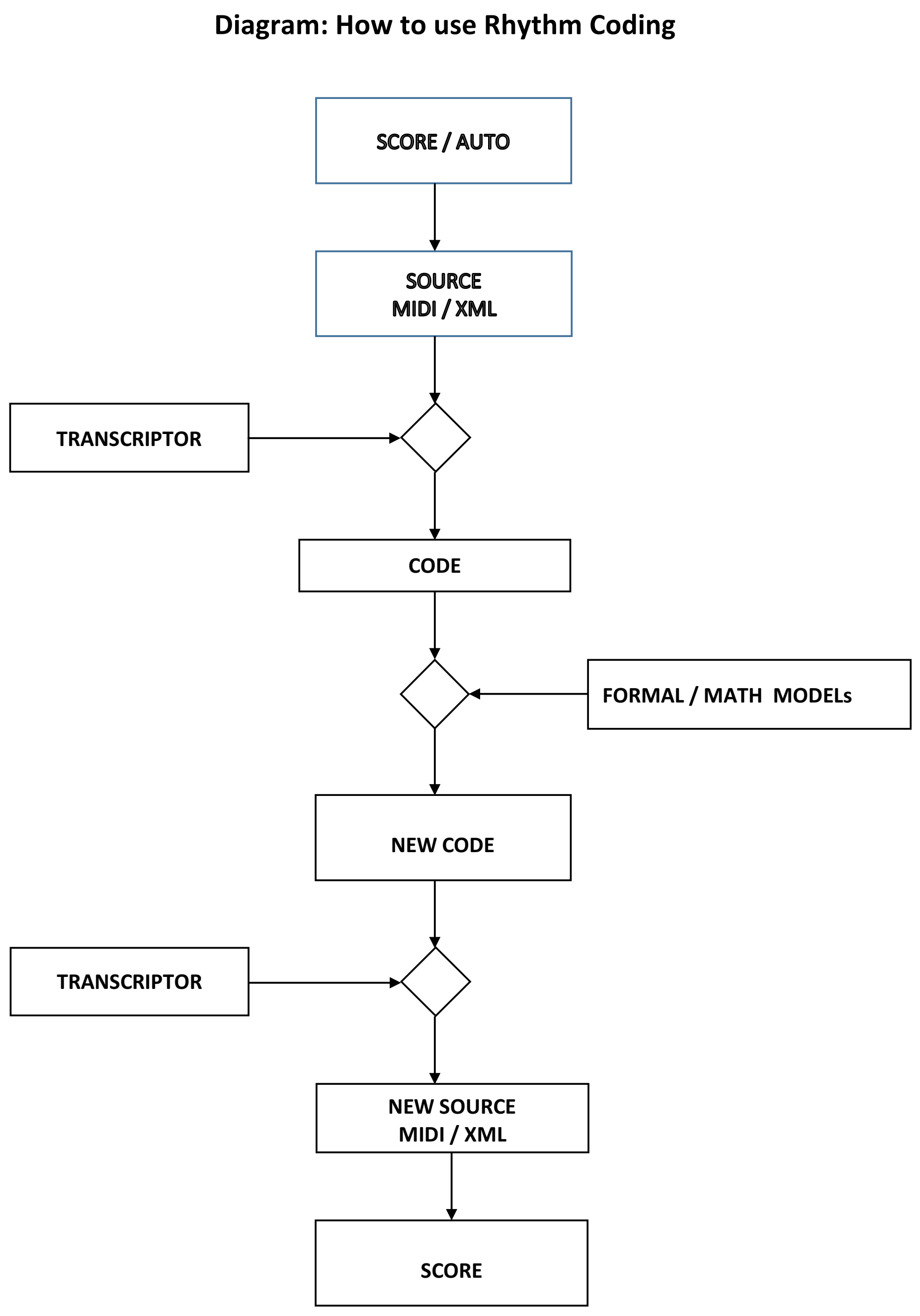


Figure 17: Diagram for Using the Extended Code.

6. Conclusions

The approach to rhythm encoding presented here is far to be complete. Many other aspects of rhythm, mainly those related to expressiveness such as rubato, grace notes among others, are not covered by the notations we show here. However, the principles of encoding we introduce along the exposition are quite general. In addition, there are, of course, many other functions and operations which can be formally applied to rhythm patterns. This strongly depends on the creativeness of the composer and most probably it would be necessary to extend the alphabet, besides rules and operations, in order to code new patterns. A great number of new Rhythm Morphologies can be derived which can be used for analysis and composition as, for example, polyrhythm patterns, written as a matrices of rhythm codes. So, we hope this work can help students and researchers as a hint for creative musical work using this mathematical approach to explore many different Rhythm Spaces.

References

1. Boenn, Georg. 2018. *Computational Models of Rhythm and Meter*, Springer Int. Pub. AG.
2. Cage, John. 1969. *Notations*, Something Else Press Inc.
3. Cook, Nicholas. 2004. Computational and Comparative Musicology. In: Clarke, Eric; Cook, Nicholas (Eds.). *Empirical Musicology, Aims, Methods, Prospects*. New York: Oxford University Press.
4. Cooper, Grosvenor; Meyer, Leonard. 1960. *The Rhythmic Structure of Music*. Chicago: University of Chicago Press.
5. Crochemore, Maxime; Hancart, Christophe; Lecroq, Thierry. 2007. *Algorithms on Strings*. Cambridge: Cambridge University Press.
6. Crochemore, Maxime; Rytter, Wojciech. 2002. *Jewels of Stringology*. Singapore: World Scientific Publishing Co. Pte. Ltda.
7. Gould, Elaine. 2011. *Behind Bars, The Definitive Guide to Music Notation*. Los Angeles: Alfred Music Publishing.
8. Hook, Julian. 1998. Rhythm in the Music of Messiaen: an Algebraic Study and an Application in the *Turangalîla Symphony*. *Music Theory Spectrum*, v. 20, n. 1, p. 97–120.

9. Sethares, William. 2007. *Rhythm and Transforms*. New York: Springer Publishing.
10. Toussaint, Godfried. 2013. *The Geometry of Musical Rhythm: What Makes a ''Good" Rhythm Good?* Florida: CRC Press.
11. Varèse, Edgard; Chou, Wen-chung. 1996. The Liberation of Sound. *Perspectives of New Music*, v. 5, n. 1, p. 11–19.