A GEOMETRIC LANGUAGE FOR REPRESENTING STRUCTURE IN POLYPHONIC MUSIC

David Meredith
Aalborg University, Denmark
dave@create.aau.dk

ABSTRACT

In 1981, Deutsch and Feroe proposed a formal language for representing melodic pitch structure that employed the powerful concept of hierarchically-related pitch alphabets. However, neither rhythmic structure nor pitch structure in polyphonic music can be adequately represented using this language. A new language is proposed here that incorporates certain features of Deutsch and Feroe's model but extends and generalises it to allow for the representation of both rhythm and pitch structure in polyphonic music. The new language adopts a geometric approach in which a passage of polyphonic music is represented as a set of multidimensional points, generated by performing transformations on component patterns. The language introduces the concept of a periodic *mask*, a generalisation of Deutsch and Feroe's notion of a pitch alphabet, that can be applied to any dimension of a geometric representation, allowing for both rhythms and pitch collections to be represented parsimoniously in a uniform way.

1. INTRODUCTION

The problem addressed here is that of designing a formal *coding language* [9, p. 155] for precisely and parsimoniously describing structure in polyphonic music. In general, there are various ways in which a musical pattern can be perceived to be constructed, and a music coding language should be capable of representing these different interpretations. Moreover, it should be possible to compare and evaluate encodings of the different ways of interpreting a musical pattern. The various methods that have been proposed for carrying out such comparisons and evaluations fall into two categories: those based on the *likelihood principle* of preferring the most *probable* interpretations; and those based on the *minimum principle* of preferring the *simplest* interpretations [9, p. 152]. Typically, statistical approaches to musical structure analysis (e.g., [8]) apply the likelihood principle, whereas approaches in the tradition of Gestalt psychology (e.g., [1]) apply the minimum principle. Indeed, van der Helm and Leeuwenberg [9, p. 153] suggest that the fundamental principle of Gestalt psychology, Koffka's [2] law of *Prägnanz*, which favours the simplest and most stable interpretation, can be seen as an "ancestor" of the minimum principle.

The language proposed here is in the tradition of the Gestalt-based coding languages for music proposed by Simon and Sumner [7], Restle [5] and Deutsch and Feroe [1]. It attempts to generalise and extend earlier languages by adopting a geometric approach, along the lines of that proposed by Meredith *et al.* [4]. A passage of polyphonic music is represented in the proposed language as a set of multidimensional points, generated by performing geometric transformations on component patterns. The language introduces the concept of a periodic *mask*, a generalisation of Deutsch and Feroe's notion of a pitch alphabet, that can be applied to any dimension of a geometric representation, allowing for both rhythms and pitch collections to be represented parsimoniously in a uniform way.

A coding language of the type proposed here could be used in music information retrieval to allow for the discovery of patterns in a database that relate to a query pattern on a deeper structural level than the surface. For example, two patterns might be perceived to be related because they have a similar structure but use different pitch alphabets (e.g., where a melody is repeated in a different mode) or because they have the same pitch interval structure but different rhythms. If the music is first encoded in a way that represents these structures, then such relationships can be automatically discovered more efficiently.

2. BACKGROUND

The earliest coding language for music was proposed in 1968 by Simon and Sumner [7]. Simon and Sumner recognized that music is multidimensional and aimed to allow for the description of patterns in melody, harmony, rhythm and form. Their language treats each of these dimensions as an independent series of symbols, chosen from alphabets, for which a compact representation can be derived in terms of certain basic element operators, such as SAME and NEXT. Simon and Sumner defined the notion of a cyclic alphabet and recognized the usual tonal scales and chords as "common" pitch alphabets that are "ordered sets already defined in the culture".

In his experiments on serial pattern learning, Restle [5] found that subjects are particularly good at identifying "runs" (e.g., (2 3 4 5)) and "trills" (e.g., (5 4 5 4)) and

tend to use these to segment a series of symbols. Restle explained this by proposing that runs and trills allow for particularly simple generative descriptions. He presented an "E-I theory" wherein a rule consists of a set E of "events" (roughly equivalent to an alphabet) and a set I of "intervals" (in the musical sense). Runs are then the set of products of E-I rule systems in which I contains only one interval. Restle's language uses the sequence operators M (for mirror), T (for transposition) and R (for repeat) for producing compact descriptions of sequences. For example, if the numbers 1 to 6 represent a row of 6 lights that can either be on or off, then the sequence (1 2 1 2 2 3 2 3 6 5 6 5 5 4 5 4) can be encoded as M(T(R(T(1)))). Restle represented structures in his language as hierarchical trees and presented an analysis of the theme of Bach's first Two-part Invention (BWV 772) which describes its tonal and motivic structure.

Deutsch and Feroe's [1] model is in the tradition of the serial pattern languages proposed by Restle [5], Leeuwenberg [3] and Simon [6]. The language can be used to encode arhythmic monophonic sequences of pitches (i.e., neither polyphony nor rhythm can be encoded). *Structures* are defined to be sequences of the elementary operators 'same' (s), 'next' (n) and 'predecessor' (p), that operate over *alphabets*, which are linearly ordered sets of symbols. Structures are decoupled from alphabets. A *sequence*, $\{S; \alpha\}$, is the application of a structure S to an alphabet, α; and a "sequence of notes" (actually a sequence of pitches) can be generated by applying a sequence to a *reference element*. *Compound sequences* can be built up from sequences by using the sequence operators, 'prime' (pr), 'retrograde' (ret), 'inversion' (inv) and 'alternation' (alt). Alphabets can be defined as sequences and 'stacked' hierarchically. For example, the C major scale would be defined as a subset of the chromatic scale (denoted by 'Cr'), using the expression $\mathrm{C} = \{\{(*, 2n^2, n, 3n^2, n); \mathrm{Cr}\}; \mathrm{c}\}$ and the C major triad could be defined relative to the C major scale by the expression $\{\{(*, 2n^2, n^3); \mathrm{C}\}; 1\}$.

The music encoding language presented in the next section extends and generalises these notions of structures and alphabets by re-casting them in geometric terms.

3. A GEOMETRIC MUSIC ENCODING LANGUAGE

In the language proposed here, a passage of music is represented as a set of *notes*. A *note*, n, is an ordered pair, $n = \langle t, p \rangle$, where $t = \mathrm{t}(n)$ is the *onset time* of the note in tatums and $p = \mathrm{p}(n)$ is the note's MIDI note number. A note is therefore a point in *note space* which is the two-dimensional Euclidean integer lattice in which the x co-ordinate represents time in tatums and the y co-ordinate represents pitch in terms of MIDI note number.

A *vector*, v, is an ordered 4-tuple, $v = \langle t, p, \mathbf{M}_\mathrm{t}, \mathbf{M}_\mathrm{p} \rangle$, where $t = \mathrm{t}(v)$ is the *time component* of v, $p = \mathrm{p}(v)$ is the *pitch component* of v, $\mathbf{M}_\mathrm{t} = \mathbf{M}_\mathrm{t}(v)$ is the *time mask sequence* of v and $\mathbf{M}_\mathrm{p} = \mathbf{M}_\mathrm{p}(v)$ is the *pitch mask sequence* of v. $\mathrm{t}(v)$ and $\mathrm{p}(v)$ are integers. $\mathbf{M}_\mathrm{t}(v)$ and $\mathbf{M}_\mathrm{p}(v)$ are *mask sequences*.

```
m(m, i)
1   if i = nil return nil
2   j ← o(m), k ← 0
3   if i ≥ o(m)
4       while j < i
5           k ← k + 1
6           j ← j + s(m)[(k − 1) mod |s(m)|]
7   else
8       while j > i
9           k ← k − 1
10          j ← j − s(m)[k mod |s(m)|]
11  if j = i return k
12  return nil
```

Figure 1. The function $\mathrm{m}(m, i)$, where m is a mask and i is an integer or nil.

```
u(m, i)
1   if i = nil return nil
2   j ← o(m), k ← 0
3   if i ≥ 0
4       while k < i
5           k ← k + 1
6           j ← j + s(m)[(k − 1) mod |s(m)|]
7   else
8       while k > i
9           k ← k − 1
10          j ← j − s(m)[k mod |s(m)|]
11  return j
```

Figure 2. The function $\mathrm{u}(m, i)$, where m is a mask and i is an integer or nil.

A *mask sequence*, $\mathbf{M}$, is an ordered set of *masks*, $\mathbf{M} = \langle m_1, m_2, \ldots, m_k \rangle$. A *mask*, m, is an ordered pair, $m = \langle o, s \rangle$, where $o = \mathrm{o}(m)$ is an integer called the *offset* of the mask and $s = \mathrm{s}(m)$ is an ordered set of integers called the *structure* of the mask. Each integer in a mask structure is called an *interval*. If m is a mask and i is an integer, then the function $\mathrm{m}(m, i)$ (see Figure 1) returns the *masked value* of i for the mask m; and the function $\mathrm{u}(m, i)$ (see Figure 2) returns the *unmasked value* of i for the mask m.

Figure 3 illustrates how a mask is used to map a subset of the integers onto the complete set of integers. In the upper part of Figure 3, the mask $\langle 3, \langle 2, 2, 1, 2, 2, 2, 1 \rangle \rangle$ is applied. The mask defines a periodic repeated pattern of intervals on the number line such that successive elements in the pattern are mapped onto successive integers. For example, the mask offset, 3, is mapped onto 0. The next integer that does not map onto nil is 5, which therefore maps onto 1, and so on. This particular mask, $\langle 3, \langle 2, 2, 1, 2, 2, 2, 1 \rangle \rangle$, can be used to represent the E♭ major scale; and the structure of this mask, $\langle 2, 2, 1, 2, 2, 2, 1 \rangle$, can be used to represent the class of all major scales.

Figure 3 also illustrates that the output of one mask can be given as input to another. Thus we can take the range of the mask $\langle 3, \langle 2, 2, 1, 2, 2, 2, 1 \rangle \rangle$ and apply the function in Figure 1 to these values with the mask $\langle 4, \langle 2, 2, 3 \rangle \rangle$ to give the result shown in the lower part of Figure 3. In

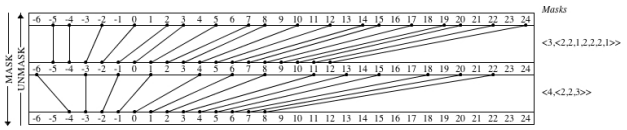

Figure 3. Applying the mask sequence, $\langle\langle 3, \langle 2, 2, 1, 2, 2, 2, 1\rangle\rangle, \langle 4, \langle 2, 2, 3\rangle\rangle\rangle$.

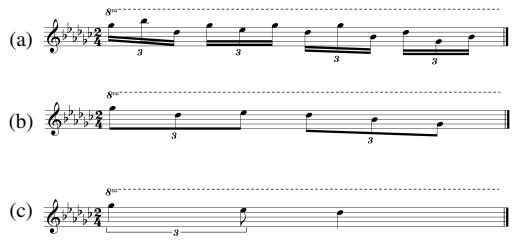

Figure 4. (a) The right-hand part of the first bar of Chopin's *Étude*, Op. 10, No. 5. (b) A plausible rhythmic reduction of (a). (c) A plausible rhythmic reduction of (b).

this particular case, the mask $\langle 4, \langle 2, 2, 3\rangle\rangle$ can be used to represent a dominant triad in a seven-note scale. Applying this to the output of the mask $\langle 3, \langle 2, 2, 1, 2, 2, 2, 1\rangle\rangle$ therefore gives a representation of the dominant triad in E♭ major—i.e., the B♭ major triad. If the topmost number line in Figure 3 represents MIDI note number, then MIDI pitch 10 maps onto 4 in the middle number line, representing the fact that 10 is the fifth scale degree in E♭ major. The number 4 in the middle line is then mapped onto 0 in the lowest number line, representing the fact that this scale degree is now the root of the dominant triad. Thus, the dominant triad in E♭ major can be represented by the mask sequence, $\langle\langle 3, \langle 2, 2, 1, 2, 2, 2, 1\rangle\rangle, \langle 4, \langle 2, 2, 3\rangle\rangle\rangle$. This example demonstrates that a mask sequence can be used to represent the notion of hierarchically related pitch alphabets proposed by Deutsch and Feroe [1]. For example, the mask sequence $\langle\langle 3, \langle 2, 2, 1, 2, 2, 2, 1\rangle\rangle, \langle 4, \langle 2, 2, 3\rangle\rangle\rangle$ corresponds to Deutsch and Feroe's alphabet,

$$\{\{(*, 2n^2, n^3); \{\{(*, 2n^2, n, 3n^2, n); \mathrm{Cr}\}; e♭\}\}; 5\} .$$

However, unlike Deutsch and Feroe's pitch alphabets, mask sequences can also be used to represent rhythmic and metric structures. For example, the crotchet metric level in a 4/4 bar in which the tatum is a semiquaver can be represented by the mask sequence $\langle\langle 0, \langle 2\rangle\rangle, \langle 0, \langle 2\rangle\rangle\rangle$. Similarly, the dotted crotchet metric level in a 6/8 bar where the tatum is a semiquaver can be represented by $\langle\langle 0, \langle 2\rangle\rangle, \langle 0, \langle 3\rangle\rangle\rangle$.

Figure 4 (a) shows the right-hand part of the first bar of Chopin's *Étude*, Op. 10, No. 5. Figure 4 (b) and (c) show plausible rhythmic reductions of the surface in Figure 4 (a). If we use the triplet semiquaver as the tatum duration, then the rhythm of Figure 4 (b) can be represented by the mask sequence $\langle\langle 0, \langle 2\rangle\rangle\rangle$; and the rhythm of Figure 4 (c) could be represented by the mask sequence $\langle\langle 0, \langle 2\rangle\rangle, \langle 0, \langle 2, 1, 3\rangle\rangle\rangle$ (or $\langle\langle 0, \langle 4, 2, 6\rangle\rangle\rangle$).

If $\mathbf{M}$ is a mask sequence and i is an integer, then the function $\mathrm{m}(\mathbf{M}, i)$ (see Figure 5) returns the masked value of i for the mask sequence $\mathbf{M}$ (i.e., the result of applying

```
m(M, i)
1  k ← i, j ← 0
2  while j < |M|
3      k ← m(M[j], k)
4      j ← j + 1
5  return k
```

Figure 5. The function $\mathrm{m}(\mathbf{M}, i)$ where $\mathbf{M}$ is a mask sequence and i is an integer.

```
u(M, i)
1  k ← i, j ← |M| − 1
2  while j ≥ 0
3      k ← u(M[j], k)
4      j ← j − 1
5  return k
```

Figure 6. The function $\mathrm{u}(\mathbf{M}, i)$ where $\mathbf{M}$ is a mask sequence and i is an integer.

each of the masks in $\mathbf{M}$, in turn, with an initial input value of i). Conversely, the function $\mathrm{u}(\mathbf{M}, i)$ in Figure 6 returns the unmasked value of i for the mask sequence $\mathbf{M}$.

If v is a vector, then $\mathbf{M}_\mathrm{t}(v)$ and $\mathbf{M}_\mathrm{p}(v)$ together define the *space*, $\mathcal{M}(v) = \langle \mathbf{M}_\mathrm{t}(v), \mathbf{M}_\mathrm{p}(v)\rangle$, in which v is defined. If a vector, v, is in note space, then the vector can be written as an ordered pair, $\langle \mathrm{t}(v), \mathrm{p}(v)\rangle$, without specifying the time and pitch mask sequences (e.g., $\langle 2, 3\rangle = \langle 2, 3, \langle\langle 0, \langle 1\rangle\rangle\rangle, \langle\langle 0, \langle 1\rangle\rangle\rangle\rangle\rangle$). Given a note n_1 and a vector v, then we can *translate* n_1 by v to give a new note, n_2. In order to do this, n_1 must first be mapped onto a point, q_1, in the space $\mathcal{M}(v)$. q_1 is then translated by v in the usual geometric way to another point, q_2, in $\mathcal{M}(v)$. Finally, q_2 is mapped back onto a note, n_2, in note space. If n is a note and v is a vector, then this process of translation can be accomplished using the algorithm in Figure 7.

Note that, if v is a vector in any space other than note space, then there will not, in general, be a unique vector in note space to which v is equivalent. For example, if $v = \langle 1, 1, \langle\langle 0, \langle 2, 3\rangle\rangle\rangle, \langle\langle 0, \langle 3, 2\rangle\rangle\rangle\rangle$, then Figure 8 shows that there are 4 distinct vectors in note space to which v might be equivalent, depending on the note that is being translated. A consequence of this is that, in general, there is no unique vector in any space that is equivalent to the sum of two or more vectors that are in different spaces. The

```
T(n, v)
1  x1 ← m(Mt(v), t(n))
2  y1 ← m(Mp(v), p(n))
3  x2 ← x1 + t(v)
4  y2 ← y1 + p(v)
5  t2 ← u(Mt(v), x2)
6  p2 ← u(Mp(v), y2)
7  return ⟨t2, p2⟩
```

Figure 7. The function $\mathrm{T}(n, v)$ where n is a note and v is a vector.

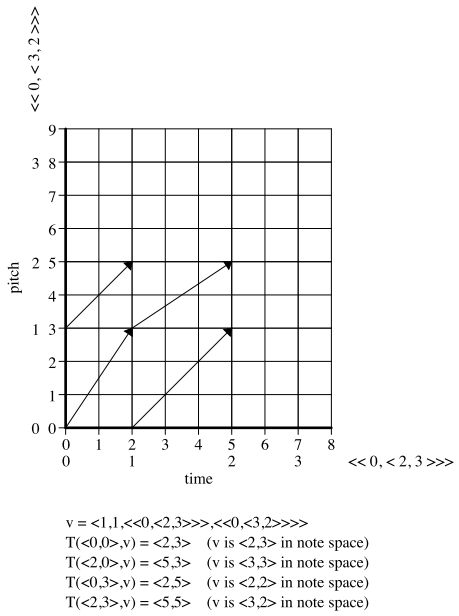

Figure 8. Equivalent vectors in note space for the vector, $v = \langle 1, 1, \langle\langle 0, \langle 2, 3\rangle\rangle\rangle, \langle\langle 0, \langle 3, 2\rangle\rangle\rangle$.

resultant vector in note space of applying two vectors in succession to a note depends on the note itself. This means that vectors in the sense defined here cannot be added together in the normal way. Instead, if a note is to be translated by the sum of two or more vectors, the vector sum has to be explicitly stated.

Since the sum of two or more vectors is not necessarily equal to a unique vector in any space, it must be considered a different type of object from a vector. We therefore define a special type of object called a *vector sum* to represent a sum of vectors. A vector sum is an *ordered set* of vectors, since vector addition in the sense defined here is not commutative. If we want to denote the sum of the vectors $v_1, v_2, \ldots v_k$, applied *in that order*, then we simply write $v_1 + v_2 + \ldots + v_k$. If w is the vector sum $v_1 + v_2 + \ldots + v_k$, then $|w| = k$, $w[j] = v_{j+1}$ and $w = \sum_{i=1}^{k} v_i$. If w_1 and w_2 are vector sums such that $w_1 = v_{1,1} + v_{1,2} + \ldots + v_{1,m}$ and $w_2 = v_{2,1} + v_{2,2} + \ldots + v_{2,n}$, then $w_1 + w_2 = v_{1,1} + v_{1,2} + \ldots + v_{1,m} + v_{2,1} + v_{2,2} + \ldots + v_{2,n}$. If v is a vector, then $\mathrm{w}(v)$ returns the vector sum that contains just v. In other words, $\mathrm{w}(v)$ typecasts a vector to a vector sum. If w is a vector sum, then we define $\mathrm{w}(w) = w$. To simplify the notation, we allow for vector sums to be added to vectors without explicit typecasting. Thus if v is a vector and w is a vector sum, then $v + w = \mathrm{w}(v) + w$ and $w + v = w + \mathrm{w}(v)$.

As an example, suppose we have three mask sequences and two vectors as follows

$$\begin{aligned}
\mathbf{M}_0 &= \langle\langle 0, \langle 1\rangle\rangle\rangle \,, \\
\mathbf{M}_1 &= \langle\langle 0, \langle 2, 2, 1, 2, 2, 2, 1\rangle\rangle\rangle \,, \\
\mathbf{M}_2 &= \langle\langle 0, \langle 2, 2, 1, 2, 2, 2, 1\rangle\rangle, \langle 0, \langle 2, 2, 3\rangle\rangle\rangle \,, \\
v_1 &= \langle 1, 1, \mathbf{M}_0, \mathbf{M}_1\rangle \text{ and} \\
v_2 &= \langle 1, 1, \mathbf{M}_0, \mathbf{M}_2\rangle \,.
\end{aligned}$$

```
T(n, w)
1  n2 ← n
2  for i ← 0 to |w| − 1
3      n2 ← T(n2, w[i])
4  return n2
```

Figure 9. The function $\mathrm{T}(n, w)$ where n is a note and w is a vector sum.

```
S(V)
1  if V = ⟨⟩ return {}
2  w ← w(V[0])
3  W ← {w}
4  for i ← 1 to |V| − 1
5      w ← w + V[i]
6      W ← W ∪ {w}
7  return W
```

Figure 10. Function for computing the equivalent vector sum set, W, for $\mathcal{V}$, where $\mathcal{V}$ is either a vector sequence or a vector sum sequence.

If we now translate the note $\langle 0, 0\rangle$ by v_1, then we get the note $\langle 1, 2\rangle$, that is $\mathrm{T}(\langle 0, 0\rangle, v_1) = \langle 1, 2\rangle$. However, we cannot then translate $\langle 1, 2\rangle$ by v_2 because $\langle 1, 2\rangle$ is not in the space in which v_2 is defined. If n is a note and w is a vector sum, then the function $\mathrm{T}(n, w)$ in Figure 9 can be used to compute the note that results when n is translated by w. $\mathrm{T}(\langle 0, 0\rangle, v_1 + v_2)$ is therefore undefined, whereas $\mathrm{T}(\langle 0, 0\rangle, v_2 + v_1) = \langle 2, 5\rangle$, illustrating the fact that vector addition is not commutative in this context.

A *vector sequence* is an ordered set of vectors and a *vector sum sequence* is an ordered set of vector sums. For any given vector sequence or vector sum sequence there exists an equivalent *vector sum set* which can be computed using the function in Figure 10. Any run of k identical vectors, v, in a vector sequence can be encoded as kv. For example, $\langle v_1, 3v_2, v_3\rangle = \langle v_1, v_2, v_2, v_2, v_3\rangle$. A run of k identical vector sums, w, in a vector sum sequence can similarly be encoded as kw.

If $\mathcal{V}$ is a vector set, vector sum set, vector sequence or vector sum sequence, then the function $\mathrm{W}(\mathcal{V})$, defined in Figure 11 returns the vector sum set that is equivalent to $\mathcal{V}$.

```
W(V)
1  if V is a vector sequence or vector sum sequence
2      W ← S(V)
3  else if V is a vector set
4      W ← ∅
5      for each v ∈ V
6          W ← W ∪ {w(v)}
7  else
8      W ← V
9  return W
```

Figure 11. Function for computing the equivalent vector sum set, W, for $\mathcal{V}$, where $\mathcal{V}$ is a vector sequence, a vector sum sequence, a vector set or a vector sum set.

```
T(n, V)
1   W ← W(V)
2   N ← ∅
3   for each w ∈ W
4       N ← N ∪ {T(n, w)}
5   return N
```

Figure 12. Function for translating a note, n, by a vector set, vector sequence, vector sum set or vector sum sequence, $\mathcal{V}$.

```
T(N, V)
1   N2 ← ∅
2   for each n ∈ N
3       N2 ← N2 ∪ T(n, V)
4   return N2
```

Figure 13. Function for translating a note set, N, by a vector set, vector sequence, vector sum set or vector sum sequence, $\mathcal{V}$.

A single note, n, or a set of notes, N, can be used to generate a set of notes by translating it by a vector set, a vector sum set, a vector sequence or a vector sum sequence. Figures 12 and 13 show functions for carrying out these types of translation. A vector sum set therefore acts as a generalisation of Deutsch and Feroe's concept of a "structure". Moreover, the combination of a note and a vector sum set to generate a set of notes is a generalisation of Deutsch and Feroe's notion of combining a structure with a reference pitch to generate a sequence of pitches.

The function $\mathrm{P}(\mathbf{X})$, defined in Figure 14 is a generalisation of Deutsch and Feroe's "prime" sequence operator, "pr". The single argument, $\mathbf{X}$, is an ordered set in which each element is a vector set, vector sum set, vector sequence or vector sum sequence. The first step in $\mathrm{P}(\mathbf{X})$ is to convert each element $\mathbf{X}[i]$ into its equivalent vector sum set (lines 1–3). The zero vector sum is then included in each vector sum set, $\mathbf{Y}[i]$ (lines 4–6). $\mathrm{P}(\mathbf{X})$ then returns a set containing a vector sum for each element of the n-ary Cartesian product of the vector sum sets in $\mathbf{Y}$ (lines 7–14). Note that if $\mathbf{A}$ and $\mathbf{B}$ are sequences such that $\mathbf{A} = \langle a_1, a_2, \ldots, a_m \rangle$ and $\mathbf{B} = \langle b_1, b_2, \ldots, b_n \rangle$ then $\mathbf{A} \oplus \mathbf{B} = \langle a_1, a_2, \ldots, a_m, b_1, b_2, \ldots, b_n \rangle$.

Deutsch and Feroe's "ret" and "inv" sequence operators correspond to reflection in the geometric language proposed here: "ret" corresponds to reflection in an axis parallel to the pitch axis, while "inv" corresponds to reflection in an axis parallel to the time axis. The functions,

$$\mathrm{R_p}(v) = \langle \mathrm{t}(v), -\mathrm{p}(v), \mathbf{M}_\mathrm{t}(v), \mathbf{M}_\mathrm{p}(v) \rangle \text{ and}$$
$$\mathrm{R_t}(v) = \langle -\mathrm{t}(v), \mathrm{p}(v), \mathbf{M}_\mathrm{t}(v), \mathbf{M}_\mathrm{p}(v) \rangle$$

reflect vectors in the time axis and pitch axis, respectively. The functions in Figure 15 can be used to reflect vector sums and the functions in Figure 16 can be used to reflect sequences or sets of vectors or vector sums. Note that there is no necessity for a function analogous to Deutsch and Feroe's "alt", because the music is represented as a

```
P(X)
1   Y ← ⟨⟩
2   for i ← 0 to |X| − 1
3       Y ← Y ⊕ ⟨W(X[i])⟩
4   v0 ← ⟨0, 0⟩
5   for i ← 0 to |Y| − 1
6       Y[i] ← Y[i] ∪ {w(v0)}
7   W1 ← Y[0]
8   for i ← 1 to |Y| − 1
9       W2 ← ∅
10      for each w1 ∈ W1
11          for each w2 ∈ Y[i]
12              W2 ← W2 ∪ {w1 + w2}
13      W1 ← W2
14  return W1
```

Figure 14. The function $\mathrm{P}(\mathbf{X})$ where $\mathbf{X}$ is an ordered set in which each element is a vector set, vector sum set, vector sequence or vector sum sequence.

```
Rp(w)
1   w2 ← w(Rp(w[0]))
2   for i ← 1 to |w| − 1
3       w2 ← w2 + Rp(w[i])
4   return w2

Rt(w)
1   w2 ← w(Rt(w[0]))
2   for i ← 1 to |w| − 1
3       w2 ← w2 + Rt(w[i])
4   return w2
```

Figure 15. Functions for reflection. w is a vector sum.

set of geometrical *points* rather than a *sequence* of *symbols*. There is also no need for a scaling function to represent augmentation or diminution, since this can be accomplished using appropriate time mask sequences.

4. EXAMPLES

Figure 17 shows one of the examples used by Deutsch and Feroe [1, p. 504] to illustrate their model. This pattern can be encoded as $\mathrm{T}(n_1, \mathrm{P}(\langle 3v_1 \rangle, \langle v_2 \rangle))$ where

$$\begin{aligned} n_1 &= \langle 1, 60 \rangle \,, \\ v_1 &= \langle 1, 1, \mathbf{M}_1, \mathbf{M}_2 \rangle \,, \\ v_2 &= \langle -1, -1 \rangle \,, \\ \mathbf{M}_1 &= \langle \langle 1, \langle 2 \rangle \rangle, \langle 0, \langle 3 \rangle \rangle \rangle \text{ and} \\ \mathbf{M}_2 &= \langle \langle 0, \langle 2, 2, 1, 2, 2, 2, 1 \rangle \rangle, \langle 0, \langle 2, 2, 3 \rangle \rangle \rangle \,. \end{aligned}$$

Given that the function, $\mathrm{U}(S_1, S_2, \ldots S_k)$ returns the union of sets $S_1, S_2, \ldots S_k$, then the pattern in Figure 4 can be encoded as follows:

$$\mathrm{U}(\mathrm{T}(n_1, \mathrm{P}(\langle v_1 \rangle, \langle v_2 \rangle)), \mathrm{T}(n_2, \mathrm{P}(\langle v_3 \rangle)), \mathrm{T}(n_3, \mathrm{P}(\langle 2v_1 \rangle, \langle v_2 \rangle)))$$

$\mathrm{R_p}(\mathcal{V})$
1 $W_1 \leftarrow \mathrm{W}(\mathcal{V})$
2 $W_2 \leftarrow \emptyset$
3 **for each** $w \in W_1$
4 $W_2 \leftarrow W_2 \cup \{\mathrm{R_p}(w)\}$
5 **return** W_2

$\mathrm{R_t}(\mathcal{V})$
1 $W_1 \leftarrow \mathrm{W}(\mathcal{V})$
2 $W_2 \leftarrow \emptyset$
3 **for each** $w \in W_1$
4 $W_2 \leftarrow W_2 \cup \{\mathrm{R_t}(w)\}$
5 **return** W_2

Figure 16. Functions for reflection. $\mathcal{V}$ is a vector set, vector sum set, vector sequence or vector sum sequence.

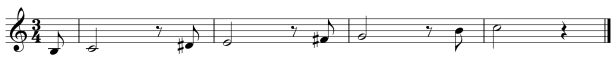

Figure 17. Example pattern used by Deutsch and Feroe [1, p. 504].

where

$$
\begin{aligned}
n_1 &= \langle 0, 90\rangle \ , \\
n_2 &= \langle 4, 87\rangle \ , \\
n_3 &= \langle 6, 85\rangle \ , \\
v_1 &= \langle 1, -1, \mathbf{M}_1, \mathbf{M}_2\rangle \ , \\
v_2 &= \langle 1, 1, \mathbf{M}_3, \mathbf{M}_2\rangle \ , \\
v_3 &= \langle 1, 1, \mathbf{M}_3, \mathbf{M}_4\rangle \ , \\
\mathbf{M}_1 &= \langle\langle 0, \langle 2\rangle\rangle\rangle \ , \\
\mathbf{M}_2 &= \langle m_1, \langle 0, s_1\rangle\rangle \ , \\
\mathbf{M}_3 &= \langle\langle 0, \langle 1\rangle\rangle\rangle \ , \\
\mathbf{M}_4 &= \langle m_1, \langle 3, s_1\rangle\rangle \ , \\
m_1 &= \langle 6, \langle 2, 2, 1, 2, 2, 2, 1\rangle\rangle \text{ and} \\
s_1 &= \langle 2, 2, 3\rangle \ .
\end{aligned}
$$

Figure 18 shows bars 320–322 from the first movement of Beethoven's Sonata in E♭, Op. 7. This passage can be encoded as $\mathrm{U}(A, B)$ where

$$
\begin{aligned}
A &= \mathrm{T}(n_1, \mathrm{P}(\langle 2v_1\rangle, \langle 5v_2\rangle)) \ , \\
B &= \mathrm{T}(n_2, \mathrm{P}(\langle 17\langle 1, 0\rangle\rangle)) \ , \\
n_1 &= \langle 0, 65\rangle \ , \\
n_2 &= \langle 0, 58\rangle \ , \\
v_1 &= \langle 0, 1, \mathbf{M}_1, \mathbf{M}_2\rangle \ , \\
v_2 &= \langle 1, -1, \mathbf{M}_1, \mathbf{M}_2\rangle \ , \\
\mathbf{M}_1 &= \langle 0, \langle 3\rangle\rangle \text{ and} \\
\mathbf{M}_2 &= \langle\langle 3, \langle 2, 2, 1, 2, 2, 2, 1\rangle\rangle, \langle 6, \langle 2, 2, 3\rangle\rangle\rangle \ .
\end{aligned}
$$

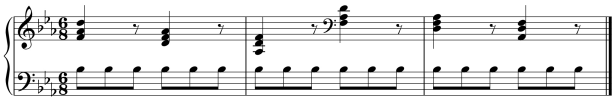

Figure 18. Bars 320–322 of Beethoven's Sonata in E♭, Op. 7, 1st. mvt.

5. CONCLUSIONS AND FUTURE WORK

This paper has introduced a geometric coding language for describing musical structure that extends Deutsch and Feroe's [1] model and recasts it in geometrical terms, allowing rhythmic and pitch structure in polyphonic music to be expressed as transformations on sets of points. The language introduces the concepts of masks, mask sequences and masked spaces which generalise Deutsch and Feroe's notion of hierarchical alphabets to the time dimension, allowing rhythms and pitch collections to be represented parsimoniously in a uniform way. A Java implementation of the language and some extended encoding examples are freely available online. [1]

The primary design goal of the language described here is that it should allow for the formulation of minimal-length descriptions of musical works. There are many ways in which the language could be developed further. For example, decoupling vector co-ordinate values from spaces could permit repetitions of vector co-ordinate value patterns in different spaces to be represented more parsimoniously. There are also cases where structure might be expressed more compactly if pitch class were decoupled from pitch height. Such decoupling of information types and other potentially useful modifications will be explored in the near future. Longer-term goals include

- to develop algorithms for automatically inferring encodings from note sets,
- to develop appropriate measures for description length and
- to explore the relationship between such an encoding language and the way that musical structure is represented cognitively and neurologically.

6. REFERENCES

[1] D. Deutsch and J. Feroe. The internal representation of pitch sequences in tonal music. *Psychol. Rev.*, 88(6):503–522, 1981.

[2] K. Koffka. *Principles of Gestalt Psychology*. Harcourt Brace, New York, 1935.

[3] E. L. J. Leeuwenberg. A perceptual coding language for visual and auditory patterns. *Am. J. Psychol.*, 84(3):307–349, 1971.

[4] D. Meredith, K. Lemström, and G. A. Wiggins. Algorithms for discovering repeated patterns in multidimensional representations of polyphonic music. *J. New Music Res.*, 31(4):321–345, 2002.

[5] F. Restle. Theory of serial pattern learning: Structural trees. *Psychol. Rev.*, 77(6):481–495, 1970.

[6] H. A. Simon. Complexity and the representation of patterned sequences of symbols. *Psychol. Rev.*, 79(5):369–382, 1972.

[7] H. A. Simon and R. K. Sumner. Pattern in music. In B. Kleinmuntz, editor, *Formal representation of human judgment*. Wiley, New York, 1968.

[8] D. Temperley. *Music and Probability*. MIT Press, Cambridge, MA., 2007.

[9] P. A. van der Helm and E. L. J. Leeuwenberg. Accessibility: A criterion for regularity and hierarchy in visual pattern codes. *J. Math. Psychol.*, 35:151–213, 1991.

[1] Java code available at `http://tinyurl.com/bl7rfgd`, longer examples available at `http://tinyurl.com/blqkgmq`.