

Atomic Operations for Algorithmic Composition

Michael Gogins
gogins@pipeline.com

December 31, 2006

I consider certain mathematical properties of score generating operations that should prove useful in algorithmic composition, especially orthogonality and the definition of metric spaces that closely reflect some basic elements of musical semantics. I define some spaces in which, as far as possible, the treatment of voices is completely orthogonal to the treatment of chords — which is not the case, for example, in the *Tonnetz* or in traditional music theory.

1 Introduction

I seek a minimal and orthogonal set of atomic operations that define a metric space which closely reflects musical semantics, for generating scores in the sense of sequences of chords as functions of time, where melody, arpeggiation, and counterpoint are represented as sequences of more or less fleeting chords. My objective is to locate a purely mathematical form of musical abstraction sufficiently above the level of single notes to be compositionally useful, sufficiently below the level of academic or pragmatic music theory to be without any inherent musical style of its own, and powerful enough to represent or generate the pitch and time structure of any score.

1.1 Definitions

I have found it necessary to make a number of distinctions not usually made, or not always made sufficiently clearly, in music theory. The following definitions describe these distinctions. Because they are logically interdependent, the definitions are listed in alphabetical order.

Atomic operation An operation that cannot be decomposed into simpler operations. In composing music, one example of an atomic operation is transposition. Whether or not an operation is atomic depends upon the space in which it is performed; alternatively, a set of atomic operations defines a space. For example, in the *Tonnetz*, chord progression is an atomic operation, performed by translating a point representing a chord to another location. But in a space that represents individual notes, chord progression is not an atomic operation, because it must be performed by separately translating each voice of the chord.

Chord A sound that is the perceptual gestalt of several notes sounding at the same time. The fundamental theoretical meaning is the *unordered* pitch-classes of a set of n tones that sound at the same time; i.e. a pitch-class set in vertical combination, not distinguishing register or inversion. However, “chord” is a polyvalent term and can mean a set of actual pitches, distinguishing octaves, ordered pitches distinguishing inversion, and so on; the precise meaning is usually clear from the context. In the present work, a chord can contain one, two, three, or more pitch-classes. Unordered pitch-classes are represented here as sets sorted in ascending order by pitch, e.g. the C major chord is $\{0, 4, 7\}$.

Chord progression A movement of voices that sends one chord to another chord; most fundamentally, a movement from one point in the *Tonnetz* to another. In the present work, this term applies to both tonal and atonal music.

Chord space An n -dimensional metric space, in which each dimension represents the pitch of one voice of a voiced chord, distinguishing inversions, octaves, and order of voices. This space extends to infinity in all directions from a reference pitch at the origin. Chord space is continuous, and every equally tempered and non-equally tempered n -voiced chord is represented as a point in the space. Note that pitches in a chord might be doubled. Music theorists have defined various equivalence classes for chord space. For each equivalence class (such as octave equivalence, permutational equivalence, inversional equivalence, transpositional equivalence), as well as for each *combination* of equivalence classes, there is a corresponding orbifold in chord space. The equivalence(s) define which points of the space are identified to form an orbifold [1].

Compositional operation An atomic operation that can be used in composing a musical score. What the atomic operations are depends upon the context. One example is moving a chord in ranged chord space to generate a voice-leading. Another example is moving a single voice to generate a melody.

Equivalence class A set whose elements are considered to be equivalent. For example, under octave equivalence, middle C and C above middle C are equivalent. In music theory, it is useful to define various equivalence classes in different contexts: octave equivalence, transpositional equivalence, inversional equivalence, and so on.

Metric space A space in which there is some consistent measure of distance between any two points. Ordinary Euclidean space is the most familiar example. Musical pitch is another example.

Minimal pitch-class set An unordered pitch-class set without duplicate pitch-classes.

Musical semantics The musical meanings, i.e. notes, chords, inversions of chords, voicings, voice-leading, melodies, and so forth, that are communicated by certain classes of sounds.

Normal chord That voicing of a chord which is closest to the orthogonal axis of the *Tonnetz*; the normal chords form a “dart” in the *Tonnetz* that can be rotated to generate the entire *Tonnetz*. Similar to, but not to be confused with, the notions of “normal form” in Larry Solomon’s atonal set theory definition, or “root position” in traditional tonal theory.

Note A sound that is a perceptual gestalt with respect to pitch and time: one pitch (or indeterminate pitch) at one time.

Orbifold A quotient space whose points are equivalent under some group action (any combination of translation, rotation, glide rotation, and reflection) that defines a symmetry. In other words, the group action permutes the corners of the orbifold, thus gluing faces of the quotient space together. For example, a strip of paper is a quotient space of the plane; gluing two ends together creates a ring, which is an orbifold. Giving the ends a half twist before joining them creates a Moebius band, another orbifold.

Orthogonal In mathematics, *perpendicular*, e.g. the dimensions of a space each of which is at right angles to all the other dimensions. In computer science, *independent*, e.g. a set of operations each of which affects only one component of a system without affecting any of the other components. Obviously, the two senses are related.

Pitch A perceptual gestalt of the logarithm of frequency. In the present work, pitch is measured, like MIDI key number, in semitones where middle C is 60; but unlike key number, pitches can be fractions, less than 0, or greater than 127.

Pitch-class The pitch of a tone *modulo* the octave, i.e. pitch under octave equivalence, i.e. $\mathbf{R}/12\mathbf{Z}$, where the octave is defined as 12 semitones.

Pitch-class set A set of pitch-classes.

Prime chord A normal chord that is equal to its own zero chord; geometrically, the prime chords form that layer of the normal chord dart that intersects the origin. This layer can be moved along the orthogonal axis of the space to generate the entire normal chord dart. Similar to, but not to be confused with, the notion of “prime form” in the sense of Forte or, more precisely, Solomon [2], since inversions are distinguished (e.g. the major triad is not the same prime chord as the minor triad).

Ranged chord space A metric space with one dimension per voice, in which each point is a voiced chord that distinguishes octaves and inversions. Ranged chord space is, of course, also an orbifold like the *Tonnetz*, but its modulus is greater than the octave: $(\mathbf{R}/R\mathbf{Z})^n$.

Tone A note of determinate pitch.

Tonnetz A chord space in which each point is an unordered pitch-class set [3], i.e. the orbifold that divides pitch-class set space for n voices by the symmetry group for n voices: $(\mathbf{R}/12\mathbf{Z})^n/\mathbf{S}_n$. Its fundamental domains are $n - 1$ dimensional simplexes. The points of the orbifold are identified by a group action that consists of a rotation (for odd dimensions) or a rotation plus a reflection (for even dimensions), i.e. by octave equivalence for pitch-classes.

Unordered pitch-class set A set of pitch-classes without respect to order; represented as sorted from lowest to highest.

Voice Something that produces a tone, or sequence of tones.

Voiced chord A chord that *does* distinguish octaves, i.e. taking into account register and inversion. In the present work, voiced chords are represented as tuples, e.g. (0, 4, 7), (60, 64, 67), and (67, 64, 72) are all voiced C major chords.

Voice-leading by pitch-classes Just what the name says. Movements of voices are represented by distance in pitch-class, not distance in pitch, i.e. a movement from one point in the *Tonnetz* to another. This is the sense of voice-leading that is closest to music theory — “voice-leading” without qualification. In the *Tonnetz*, voice-leading by pitch-classes is exactly the same operation as chord progression.

Voice-leading by pitches Just what the name says. Movements of voices are represented by distance in absolute pitch, i.e. a movement from one point in ranged chord space to another point. This is a complete representation of what happens to the pitches of the voices in a progression of voiced chords. In chord space, voice-leading by pitches is *not* identical to chord progression; but neither are the two operations completely separable (see below).

Voice-leading distance The distance, by Manhattan or Euclidean metric, between two points in the *Tonnetz*, or in ranged chord space. The closest voice-leading are usually the best according to traditional rules of voice-leading.

Zero chord A chord transposed so that its lowest pitch-class is 0.

1.2 Compositional Operations

Algorithmic composition attracts me with its power to produce, from relatively simple or concise specifications, relatively complex or elaborate scores. Ultimately, this power derives from the recursive or iterative nature of compositional algorithms.

On the one hand, algorithmic composition enables me to create music that I could not have been imagined, and far more efficiently than by mere random sampling. But on the other hand, the recursive nature of the algorithms makes it more difficult, if not impossible, for me to predict in advance the exact effect upon a score of any given change to its specification.

Before attempting to define any actual operations — or even whether we are talking about voices, notes, or chords — a minimum compositionally useful set of properties of music to control with atomic operations must include:

- Instrument choice
- Time of onset
- Duration
- Pitch
- Chord
- Voicing
- Voice-leading
- Loudness

Note that there already is some ambiguity or overlap in these properties. A voice can simply be a voice, or it can be part of a chord. That is because the properties do not derive from acoustics, or from mathematics, but rather from a mixture of perception, compositional practice, and music theory.

My objective is to separate out and refine the purely theoretical aspects of compositional practice, so that they can be formalized and used for algorithmic composition. This can be done by developing a system of atomic mathematical operations that correspond, in an intelligible way, to atomic operations of musical composition. For the system to be useful, the atomic operations must:

- Represent atomic operations of musical composition, in a simple and intelligible way.
- Operate in a *metric space*, guaranteeing that one can use ordinary arithmetic to actually perform the operations.
- Be *orthogonal*, guaranteeing that one can create or modify one operation without, however, affecting the results of another operation; in other words, e.g., that one can perform an operation to control the horizontal movement of voices without changing their vertical combinations in chords.
- Be *complete*, guaranteeing that one can perform any compositional operation whatsoever using an atomic operation or some combination of atomic operations.

Such a system would be universal — capable of generating any possible score — and could be adapted to any number of compositional materials, algorithms, and styles.

Why must the operations be orthogonal? Because the compositional algorithms and operations I have used so far are not completely orthogonal, so that whenever I have managed to create a desired effect, that all too often has come at the cost of destroying some other, equally desired effect.

In traditional composition this is not a problem, but in algorithmic composition the recursive application of atomic operations causes them to have global effects that are computationally irreducible — their effects cannot be intuited without actually applying the algorithm. Obviously, if an operation is not orthogonal, applying it recursively will spread its unintended side effects quite unpredictably throughout the score.

Why is voice-leading by pitches not orthogonal to chord progression? On the one hand, for each chord progression there are, depending upon the range allowed the voices, several or even many voice-leading by pitches. On the other hand, if one distinguishes voice-leading only by voice-leading distance, then often there are several different chord progressions at the same voice-leading distance. So to *some* extent, horizontal movement and vertical combination are independent while, at the same time, obviously there are many *more* voice-leading to *different* chords. Mathematically speaking, therefore, the horizontal movement and the vertical combination of voices are only *partially separable*.

This inherent entanglement of counterpoint with harmony is, of course, a mathematical precondition for having any sort of interesting polyphonic music in the first place. Indeed, the *Tonnetz* itself is precisely that quotient of chord space in which voice-leading and chord progressions are identical — i.e. where there exists a one-to-one correspondence between voice-leading and chord progressions.

However, I must repeat, a set of completely orthogonal operations would *enormously* facilitate the iterative refinement of specifications for scores in algorithmic composition. And I believe that as long as a chosen set of orthogonal operations is musically intelligible, then composers can learn to think in terms of those operations instead of, or in addition to, the traditional ones.

2 Orthogonality

Operations upon instrument choice, time of onset, duration, and loudness already are both orthogonal and metric. Therefore, let us focus on the *non-orthogonal* notions of chord, voicing, and voice-leading, and see if we can take them apart and put them back together into a (perhaps slightly larger) set of truly orthogonal operations of equal power.

Our starting point is a set of related orbifolds that model symmetries of both voice-leading and harmonic progression. Each orbifold is generated by applying a different set of equivalence classes to chord space. For each orbifold, there is a ring in which we can do arithmetic.

We generate the *Tonnetz* by applying octave equivalence and chord symmetry equivalence. We generate ranged chord space by applying range equivalence. We generate normal chord space by applying chord equivalence by distance from the orthogonal axis of chord space and pitch-class set equivalence. We generate prime chord space by applying normal chord equivalence and chord transposition equivalence. We generate chord transposition space by applying prime chord equivalence and range equivalence (which leaves only the orthogonal axis of chord space *modulo* its range). For a given pitch-class set, we generate voicing space by applying range equivalence.

It might seem sufficient to extend the *Tonnetz* beyond the octave, i.e. to use ranged chord space. This does suffice to represent, within a defined range of pitches, both all chord progressions and all voice-leading. Unfortunately, as we have seen, it does nothing to separate operations on vertical structure from operations on horizontal structure.

A better alternative is:

C = chord

V = voicing

This produces an orbifold with two dimensions and quotients, which could be called the *CV torus*. Another alternative is:

P = prime chord

T = transposition

V = voicing

We can assemble a super-orbifold of $\mathbf{P} \times \mathbf{T} \times \mathbf{V}$ (prime chord space \times transposition space \times voicing space) in which fundamental operations of musical composition are:

- Addition in **P** to generate prime chords,
- Addition in **T** to generate transpositions, and
- Addition in **V** to generate voicings.

Each of these is:

- Orthogonal, so that performing an operation in one orbifold does not affect any other orbifold.
- A ring, in which addition performs the fundamental operation, but multiplication is also possible.

By doing modular matrix arithmetic in $\mathbf{PTV} \times \textit{time}$, we can generate any score. Or, at least, since we have not included instrument choice or loudness, we can generate any piano-reduction type score.

3 Arithmetic

It now remains to identify the exact arithmetic of $\mathbf{PTV} \times \text{time}$. It will be matrix arithmetic in which each dimension has a different modulus reflecting that dimension's quotient of chord space.

Obviously, arithmetic on prime chords and arithmetic on transposition can be composed in the mathematical sense to produce arithmetic on normal chords, i.e. chord progressions. These operations define an orbifold with 3 dimensions and 3 quotients, which could be called the \mathbf{PTV} torus. Let us construct the arithmetic for $\mathbf{PTV} \times \text{time}$, which is a space with 4 dimensions and 3 quotients.

Transposition of pitch-classes is already an additive cyclic group *modulo* 12, \mathbf{T} .

Regarding prime chords, first consider normal chords — unordered pitch-class sets. As long as there is a finite number of equally tempered pitches in the scale of the score, then each *unordered* pitch-class set can be represented as a binary numeral c where at most there are as many digits as the m pitch-classes per octave. For example, there are $2^{12} = 4096$ unordered pitch-class sets (including the empty set) in 12-tone equal temperament. These numbers, ranging from 0 through 4095, form a monoid \mathbf{M} (for Mason [4], who seems to have invented this general scheme of numbering chords) in which $C2 \text{ modulo } 2^m$ generates a semitone transposition of chord C .

In the context of music theory, including the empty set makes sense. In the context of algorithmic composition, including the empty set in the space means that iteratively applying an operation to an element will attract it to the empty set. This is not desirable. It is better for iterating an operation to produce a cyclical orbit in the space. The group for the space can be modified to exclude the empty set. For example, I find that the chords (*not* including the empty set) can be re-numbered from 0 through 4094 to form an additive cyclic group \mathbf{C} , in which $C2 + 1 \text{ modulo } 2^m - 1$ generates a semitone transposition.

We can consider simply the prime chords by counting only those unordered pitch-class sets whose prime chords are equal to themselves. These are similar to (but not identical with) “prime forms” in Larry Solomon’s [2] sense, i.e. Forte prime forms but respecting inversion, e.g. major triads and minor triads are distinguished. In 12-tone equal temperament, there are 355 prime chords ranging from the unison, including every chord familiar and unfamiliar as well as every scale familiar and unfamiliar, to the chromatic scale. They can be ordered by \mathbf{M} to produce an additive cyclic group by index, \mathbf{P} .

Similarly, the *ordered* voicings of each unordered pitch-class set can also be represented as a numeral in a different number system, with radix equal to the number of octaves k in the range of the score and at most one digit for each of the n voices. Octaves can be added one at a time, starting with the first voice, then carrying over to the second voice if it exists, and so on for each of the voices. These voicings (*not* including the empty set) also form an additive cyclic group \mathbf{V} , in which adding $1 \text{ modulo } n^k$ generates the next voicing.

In both the \mathbf{CV} torus and the \mathbf{PTV} torus, all operations are orthogonal, and each one is numerically ordered. Therefore, as sets of operations, each torus defines a metric space. More precisely, since all operations are modular, they define orbifolds, although they are nothing like the *Tonnetz*. Any chord, any voiced chord, and any voice-leading can be produced by a sequence of these operations.

I feel that, inasmuch as musicians have a distinct tendency to distinguish the interval structure of chords ahead of their actual pitches, the \mathbf{PTV} torus is the more musically natural and intelligible of these alternatives.

Arithmetic in $\mathbf{PTV} \times \text{time}$ is powerful enough to generate any “un-orchestrated”, piano reduction-style score in any system of equal temperament, including any and all voice-

leadings. The space falls short of being powerful enough to represent or generate any score because it does not bring operations assigning different instruments to different voices, or different loudnesses to different voices, into the space, nor can it handle non-equally-tempered scales.

The operation of finding the closest voice-leading between two chords is not a generating operation of the space, because it acts upon two points in the space at once. However, of course, it remains a very useful operation that should be available in any algorithmic composition system.

After working with $\mathbf{PTV} \times \textit{time}$ and comparing its results with those of with the many note-by-note generators that I have used in the past, I find that neither approach is quite complete. I find no obvious representation of grace notes, passing tones, arpeggiation, or melody within $\mathbf{PTV} \times \textit{time}$. At the same time, there is no question but that $\mathbf{PTV} \times \textit{time}$ has proved an effective means of generating large-scale pitch structures, with reasonable voice-leading, that the note-by-note generators have completely lacked.

Is it possible to combine the two approaches?

To a certain extent it is possible — by laying a sequence of operations in $\mathbf{PTV} \times \textit{time}$ over a sequence generated by a note-by-note generator. In this context, the operations not only *generate* notes, they also can *filter* notes, i.e. either (a) pass only the notes whose pitches match those that would have been produced by \mathbf{PTV} arithmetic at that time in the score, or (b) actively adjust the pitches of the notes to match.

$\mathbf{PTV} \times \textit{time}$, with or without filtering, is the best that I have been able to do, to date, towards my goal of locating a level of musical abstraction sufficiently above the level of single notes to be compositionally useful, and sufficiently below the level of academic or pragmatic music theory to be without any inherent musical style of its own.

4 Implementation

Experience suggests that context-free Lindenmayer systems provide an easy to implement, yet musically useful compositional algorithm for experimenting with operations and spaces. The atoms of the Lindenmayer system can include commands that perform the atomic operations in question, within the metric space or spaces that they define.

Once the operations have been refined and understood by composing with them, they can be used in other mathematical systems.

Experience also indicates that some of these operations, especially voice-leading, are computationally expensive and that both the algorithm and the code should be extensively optimized.

4.1 Lindenmayer System

Lindenmayer systems are recursive functions that rewrite strings [5]. Each Lindenmayer system consists of an axiom or initial string of atoms, a table of rules each specifying how one atom is to be replaced with a string of atoms, an implicit rule that an atom with no replacement is replaced by itself, and the number of iterations for the recursion. In addition, some of the atoms of the system are commands for a “turtle” in a turtle graphics-like system. For example, **F** might mean move one step while drawing a line, **f** move one step without drawing a line, **+** turn right, **-** turn left, **[** push the turtle state onto a stack, and **]** pop the turtle state from the stack (pushing starts a branch; popping returns the turtle to the branching point.)

P	Prime chord index.
T	Transposition by semitone.
V	Voicing index for P.
t	Time in seconds.
d	Duration in seconds.
i	Instrument number as MIDI channel.
k	Pitch of individual note as MIDI key number.
v	Loudness as MIDI velocity.

Table 1: **Dimensions**

The Lindenmayer system is iterated a the specified number times. Repeated replacements usually expand the initial axiom into a very long string of atoms, called the *production* of the system. The production is then interpreted as a program for the turtle, which draws a figure in the space.

In the simplest type of Lindenmayer system, or *OL* system, also called context-free, the replacement rules do not depend on the state of the production on either side of the current atom, and do not take parameters.

OL systems have already been used for some time to generate musical scores in spaces where time is one dimension of the space [6, 7, 8].

Table 1 summarizes the dimensions of the Lindenmayer system. In order to accomodate both chord generation, and filtering generated notes by voice-leading operations, the turtle in this system is polyvalent:

1. The turtle can be considered to be moving about in in $\mathbf{PTV} \times t$ space, either with or without voice-leading, and generates chords (**C** commands).
2. The turtle can be considered to be moving about in $tdikv$ space, and generates one note at a time (**N** commands).
3. The turtle can be considered to be moving about in $\mathbf{PT} \times t$, with voice-leading, and generates voice-leading operations that are applied to generated notes (**L** commands).

The operations on turtle step, and the assignment operations, allow much of what can be done in parametric Lindenmayer systems to be done in this non-parametric system:

$$T_i = f(T_i, S_i \times p)$$

for turtle step, and

$$S_i = f(S_i, p),$$

for step size, where

T_i = the value of the turtle state on dimension i ,

f = the operator, e.g. +,

S_i = the value of the turtle step on dimension i , and

p = the parameter of the turtle command.

Table 2 summarizes the turtle commands for the Lindenmayer system.

Chord Creation	
C=name	Set turtle chord to the prime chord of the jazz-named chord.
Ca	Write chord at current absolute position of the turtle.
Cv	Write chord with turtle's pitch-classes, but at closest voice-leading from the prior chord.
Cm	Write chord with turtle's pitch-classes, but at closest voice-leading from the prior chord, while also taking one step forward in time.
Note Creation	
N	Create note at current absolute position of the turtle; the note may be filtered by the current voice-leading operation at that time (if one exists).
Nm	Create note at current absolute position of the turtle, while also taking one step forward in time; the note may be filtered by the current voice-leading operation at that time (if one exists).
Voice-Leading Operation	
L=name	Set voice-leading operation to the prime chord and transposition of the jazz-named chord.
L	Write voice-leading operation at current absolute position of the turtle.
Lm	Write voice-leading operation at current absolute position of the turtle, while also taking one step forward in time.
Size of Turtle Step	
Smx	Multiply step by x .
Sdx	Divide step by x .
S=d, x	Assign x to dimension d of step.
S+d, x	Add x to dimension d of step.
S-d, x	Subtract x from dimension d of step.
S*d, x	Multiply dimension d of step by x .
S/d, x	Divide dimension d of step by x .
Orientation of Turtle Step	
0+a, b	Rotate turtle step by A from dimension a to b .
0-a, b	Rotate turtle step by A from dimension b to a .
Turtle Movement	
Tf	Move turtle one step.
T=d, x	Assign x to dimension d of turtle.
T+d, x	Add x to dimension d of turtle.
T-d, x	Subtract x from dimension d of turtle.
T*d, x	Multiply dimension d of turtle by x .
T/d, x	Divide dimension d of turtle by x .
Branching	
[Push turtle state on stack (branch).
]	Pop state off stack (return to branching point).

Table 2: **Turtle Commands**

4.1.1 Voice-Leading Algorithm

Voice-leading, or more precisely finding the voicing of a specified pitch-class set that has the closest voice-leading from a specified voiced chord, is by far the most computationally expensive operation.

Tymoczko introduced an algorithm for non-bijective voice-leadings [3], i.e. in which voices may be doubled to obtain closer voice-leadings. This algorithm uses dynamic programming.

Gogins introduced an algorithm for bijective voice-leadings [9], i.e. in which all chords have the same number of voices. This algorithm uses a brute-force search of all unordered chords within a certain range, but a C++ implementation of the algorithm is reasonably fast in the context of non-real-time algorithmic composition.

The Gogins algorithm is used in the present work. It is possible to double or remove voices in one of the chords of the voice-leading to ensure that it is bijective.

There is a problem in voice-leading segments of scores where there are numerous, possibly doubled, pitches. The brute-force algorithm explodes. Possible solutions include using the Tymoczko algorithm, or representing the source and target pitches by ordered pitch-class sets for the purposes of voice-leading (which is the key idea in the Tymoczko algorithm). The target pitches would then be moved to the same “inversion” as the target pitch-class set.

There is also the related problem that the arity of the target pitches can only grow with the present algorithm. This is not acceptable.

I think it is going to be necessary to do voice-leadings with ordered pitch-class sets, not pitches. I think that would solve the arity problem also.

The top and bottom voices are of course unique, but there is a potential ambiguity in the inner voices. One must go by the first encountered pitch-class set, counting up from the bottom.

References

- [1] Clifton Callender, Ian Quinn, and Dmitri Tymoczko. Generalized Chord Spaces, 2006. Unpublished, <http://music.princeton.edu/~dmitri>. 2
- [2] Larry Solomon. Solomon's music resources. <http://solomonsmusic.net>. 3, 7
- [3] Dmitri Tymoczko. The Geometry of Musical Chords. *Science*, 313:72–74, 2006. 3, 11
- [4] Robert M. Mason. An encoding algorithm and tables for the digital analysis of harmony. *Journal of Research in Music Education*, 17:286–300, 369–387, 1969. 7
- [5] Przemyslaw Prusinkiewicz and Artistid Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, New York, 1996 [1991]. Available online at <http://algorithmicbotany.org/papers>. 8
- [6] S. R. Holtzman. Using Generative Grammars for Music Composition. *Computer Music Journal*, 5(1):51–64, 1981. 9
- [7] Michael Gogins. Fractal Music with String Rewriting Grammars. *News of Music*, 13:146–170, Winter 1992. 9
- [8] Jon McCormack. Grammar Based Music Composition. In R. Stocker et al., editors, *From Local Interactions to Global Phenomena*, Complex Systems 96, Amsterdam, 1996. ISO Press. 9
- [9] Michael Gogins. Score generation in voice-leading and chord spaces. In Georg Essl and Ichiro Fujinaga, editors, *Proceedings of the 2006 International Computer Music Conference*, San Francisco, California, 2006. International Computer Music Association. 11