ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DOTTORATO DI RICERCA IN
MENTE, LOGICA E LINGUAGGIO

Settore scientifico di afferenza: M-FIL/02
Ciclo: XXIII

ON SYMBOLIC REPRESENTATIONS OF MUSIC

Presentato da
Carmine Emanuele Cella

Coordinatore: Prof. R. Nicoletti

Relatore: Prof. F. Bellissima

Correlatore: Prof. G. Corsi

Esame finale: luglio 2011

Contents

Acknowledgments

I first would like to thank to my supervisor, Prof. Fabio Bellissima, for his careful guidance during my study years and for his friendship. I'll never forget our meetings in Siena.

I also would like to thank my co-advisor, Prof. G. Corsi, for the continuous support given during these years.

I am extremely grateful to Moreno Andreatta, Carlos Agon and Gérard Assayag, for hosting me at the Musical representations team at IRCAM and to Ichiro Fujinaga for hosting me at McGill University.

Thanks to Juan Josè Burred, Philippe Esling, Arnaud Dessein and Niels Bogaards for reviewing and discussing the manuscripts and for their friendship.

Finally, I am really thankful to my great family: Bernardino, Ornella, Leonardo and Sara. Without them my life is completely meaningless; this work is dedicated to them.

Carmine Emanuele Cella
Pesaro
20 June 2011

Chapter 1

Music and representations: an introduction

Musica est exercitium arithmeticae occultum
nescientis se numerare animi.

Gottfried Leibniz

Abstract

The chapter provides an introduction to the context of this research, that is based on the essential relation between music and mathematics. After an historical overview, the main motivation of this work is explained.

Music has a dual nature. From one side, it is irrational and related to emotions. For this reason it's difficult to describe music by means of formalized languages; the process of musical creation is not linear and many steps of it are not representable by any kind of algorithmic procedure. From another side, however, music follows strict systems of rules based on formal reasoning. Many musical elements, moreover, are defined *only* through mathematics and the relation between the two fields seems to be really close.

The dichotomy created by the interpretation of music as a purely mathematical theory or as a perceptually-related field is central to the development of musical theory; it is of philosophical nature and can be considered related to the contrast between *rationalism* and *empirism*. This dichotomy is also at the origin of the interpretation of music as an *art* appeared in late XVI century.

Following sections will give an overview of the relation between music and mathematics, in order to provide the right context for this research. At the end of this chapter, a motivation for this work will be also given through the notion of *symbolic representation*.

1.1 Mathemathics and music: an historical perspective

Despite common thinking, the link between mathematics and music has ancient roots. Many researchers and musicians, such as Douglas R. Hofstadter and Edward Rothstein, produced interesting works showing how both disciplines share, on the technical level, common attributes of abstraction and beauty. The difficulties of understanding the link between mathematics and music are mainly due to the interpretation of music as an *art* appeared in late XVI century. As a consequence of this interpretation music has been treated more in expressive terms, as a language to be handled pedagogically, losing its original scientifc connotation. The two fields are actually so closed that what is surprising is, in fact, their separation: following sections will provide a short historical overview of this astonishing link.

1.1.1 From Pythagoras to Middle Ages

Even the simpler musical activity, not influenced by education or by training, is based on the identification of the relationship between adiacent tones. This relationship, the distance between two musical tones, is called *interval*.

Between all the possible intervals, there is one that is very special: it's called the *octave* and represent two tones that are expressed by the ratio $2:1$. The octave is a real *class of equivalence* for tones: even an untrained listener will say that two tones at the distance of octaves are essentialy the *same*; other important intervals are the *fifth* and the *fourth*. The beginning of the relation between music and mathematics is centered on the definition of intervals.

The problem of intervals

Pythagoras of Samos was probably the first, in VI century B.C., to define a clear connection between mathematics and music. According to legend, Pythagoras heard different tones being emitted from the striking of the anvils while passing by a blacksmith's shop. He thought that some mathematical reasons should have been at the origin of this phenomenon and that these reasons could have been applied to music too: after some observations, he discovered that it was because the hammers were:

> *[. . .] simple ratios of each other, one was half the size of the first, another was $\frac{2}{3}$ the size, ...*

The legend has been proven to be false for many reasons. Nonetheless, it is true that Pythagoras elaborated a complex system of *ratios* to mathematically define musical entities, related to the length of the string of a theoretical instrument called *monochord*.

Pythagoras discovered that the intervals of octave, fifth and fourth can be expressed by simple ratios, namely $2:1$, $3:2$ and $4:3$. All intervals can be expressed, in the Pythagorean system, by means of the *tetractys*, or pyramid of dots, a geometrical figure made up of the first four numbers.

The system developed by Pythagoras will not be examined further here; suffice it to say that the connection between mathematics and music provided by Pythagoras had great fortune and lead to the inclusion of music into the *quadrivim*, a curriculum of study for liberaral arts outlined by Plato and by Martianus Capella and constituted by arithmetic, geometry, music, and astronomy.

During Middle Ages, the quadrivium assumed a relevant part of education in universities and consequently music gained a lot of development: one major advance was the establishment of *mensural notation*.

Mensural notation

Music, like mathematics, depends on a specialized system of notation based on symbols that encode information. While the origin of musical notation can be traced back to the alphabet of ancient Greek, it's during the medieval period that a series of important developments changed it significantly and led to the mensural approach.

Mensural notation is a clear example of the link that binds together mathematics and music: it's a new form of musical notation that enable musicians to precisely write any kind of rhythm and it's basically impossible to understand it without understanding medieval arithmethics. The main features of medieval musical notation are in fact directly connected to the coeval system of measures. There are two main variants of mensural notation: the french variant was probably elaborated by Philippe de Vitry (1291-1361), while the

italian variant was described from the theoretician Marchetto da Padova (1274?-1305 or 1319). While this types of notation handles both pitches and durations, the discussion below will only focus on the second aspect .

In the french system, there are five differents values for notes: *maxima, longa, brevis, semibrevis* and *minima*. Except the *minima*, each one of them can be subdivided into two or three parts (binary and ternary subdivisions). This double possibility is ambiguous: without context information, it is not possibile to understand the real value of a note. The hierarchy of the values, moreover, is centered on the *brevis*: other values can be thought as multiples or submultiples of it.

The italian system, also centered on the *brevis* , is even more complex: the first subdivision (called *tempus perfectum secundum divisionem duodenariam*) decomposes the *brevis* into three *semibreves maiores*, each one of them being decomposed into two *semibreves minores* for a total of twelve *semibreves minimae*. Other subdivisions are:

- *tempus imperfectum secundum italicos*: in which the *brevis* is decomposed into eight *semibreves minimae*;
- *tempus perfectum secundum divisionem nonariam*: in which the *brevis* is decomposed into nine *semibreves minores*;
- *tempus imperfectum secundum gallicos*: in which the *brevis* is decomposed into six *semibreves minores*.

It's worth noting that all subdivisions are duodecimal fractions, that is fractions with twelve as denominator. Like in the french system, also in the italian all the values can be subdivided into two or three parts.

Figure 1.1 shows an example of mensural notation: it depicts a page from Nicolas Gombert's *Le chant des oyseaux* (1545). Even from the basic exposition given above, it's clear that mensural notation is not an easy matter. Probably, it has been developed inside universities in Paris and in Padova and only very specialized musicians were able to use it. Why medieval theoreticians developed such a complex system? Why they used duodecimal fractions with binary and ternary subdivisions? Why the *brevis* was the central value of the system? The answers to these questions are in the coeval system of measures.

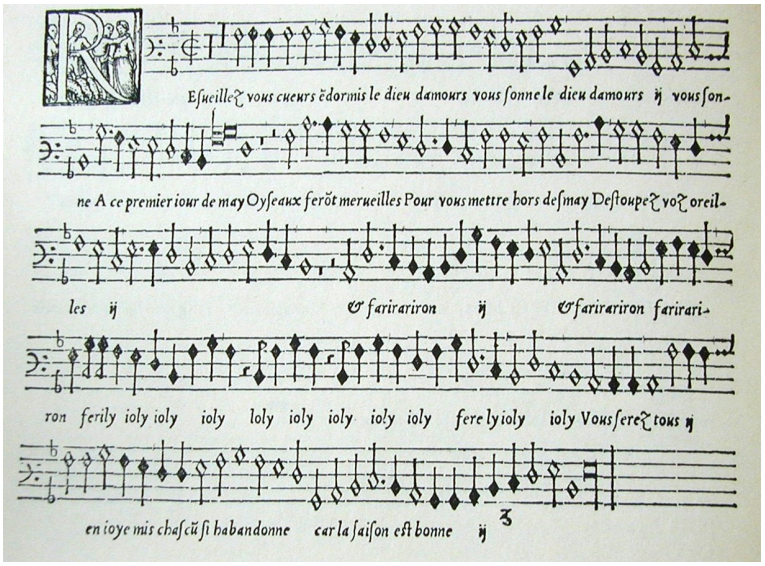


Figure 1.1: A page from Nicolas Gombert's *Le chant des oyseaux* (1545)

The medieval system of measures

Until late Middle Ages, arithmetical operations were performed using roman numbers: even if Fibonacci in his book *Liber abaci* (1202) clearly showed the supremacy of indo-arabic symbols (*cyffras)*, the Church constantly fought their usage. While roman numbers were good to perform additions and substractions, multiplications and divisions where very slow with them and were only possibly by using mnemonic tables. Operations with fractions were even worst: they required multiple passages and were sometimes impossible. A fraction like $\frac{9}{76}$ was in fact notated literally like *suboctuplasuperquadripartiens nonas* and was obviously very easy to make errors in computations.

One peculiarity of the roman system of fractions was the duodecimal base, that was also used by the medieval system of weights and by the monetary system. The bigger monetary unit was the *libra* and was possible to divide it into twelve *unciae*. Each *uncia* was divisible into *semnuciae*, that on their hand were divisible into *duellae* and so on. The measuring of time,

called *chronaca*, was also based on duodecimal fractions: the year was also called *libra* and was divided into twelve months. The hours (*horae*) were subdivided into *momenti* that were themself subdivided again into twelve *unciae*.

Common aspects

It should be clear, at this point, that the medieval system of musical notation was clearly correlated to the coeval system of measures. Both systems are based on a central value, the *brevis* and the *uncia* respectively, that can be multiplied to obtain bigger values or subdivided to have smaller ones; in this sense, both systems are organized into hierarchies.

Finally, both systems are based on values that can be subdivided by two or by three (binary and ternary subdivisions). Many other similarities exist between the two system (for one, also the graphical symbols used in mensural notation derived from coeval mathematics) but they will not be considered here.

Nonetheless it is evident that, during the Middle Ages, music and mathematics were really close each other and shared many common aspects.

1.1.2 From Rameau to Riemann

The period between XVIII and XIX century is characterized by the consolidation of the modern major/minor tonality, that will be the reference system for occidental classical music at least until the beginning of XX century. In the modern tonal system, mathematics also plays a central role. Harmonic theory of this historical period has been greatly influenced by two major theroticians: Jean-Philippe Rameau (1683-1764) and Hugo Riemann (1849-1919); both based their theories on the coeval mathematical theory.

The *corps sonore* and the physics of sound

Rameau has been a dominant figure in the musical panorama of XVIII century: on his books *Traité de l'harmonie rèduite à ses principes natuerls* (1722) and *Code de musique pratique* (1760) he defined the bases of contemporary harmonic theory.

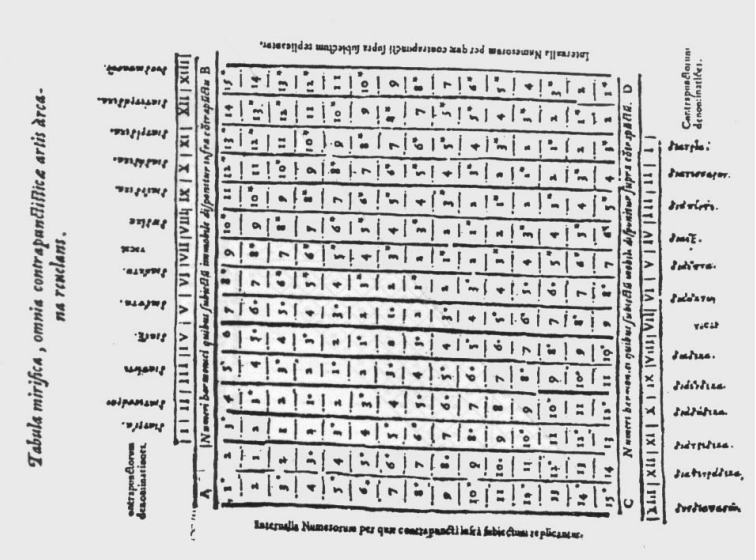


Figure 1.2: The *Tabula mirifica* by A. Kircher.

Following Pythagoras, he defined musical intervals by means of ratios derived from the division of an ideal string. He also showed an interest in practical aspects of music for the first time. While his predecessors, such as Athanasius Kircher (1602-1680), considered the monochord as a mechanical model of universal harmony (for example like the one represented in figure 1.2), Rameau considered the monochord not only an abstract tool but used it to define simultaneous sets of tones (called *chords*), throught the notion of *accord parfait*.

Rameau described, using integer ratios, the structure of major (perfect) chords calling *fundamental* the most important[1] note of the chord and numbering it as *1*. In this way he has been able to create *classes of equivalences* for chords, grouping together all chords described by the same ratios. Moreover, he defined the equivalence of the *positions* of chords, a concept that not will be examined here, showing interesting links between the rational theory of

[1]The concept of fundamental of a chord is really important in occidental music. In music theory, the fundamental (or root) of a chord is the note or pitch upon which a triadic chord is built. For example, the root of the major triad C-E-G is C.

chords and musical praxis.

Rameau didn't know, at the time he was writing his *Traitè*, the recent scientific discoveries about *upper harmonics* described by Joseph Sauveur (1653-1716). Throught vibration theory, Saveur showed that each musical tone was made of several harmonics, whose ratios correspond to the division of the string. Formally, a musical tone can be described by the sum of all integer fractions from 1 to ∞:

$$\sum_{n=1}^{\infty} \frac{1}{n} = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \dots \tag{1.1}$$

Figure 1.3 shows the so-called series of harmonics[2] from the note C corresponding to the application of equation 1.1. Each element of the summation series corresponds to a vibration *mode* of the string and the first element $(1/1)$ correspond to the *fundamental* as defined by Rameau.

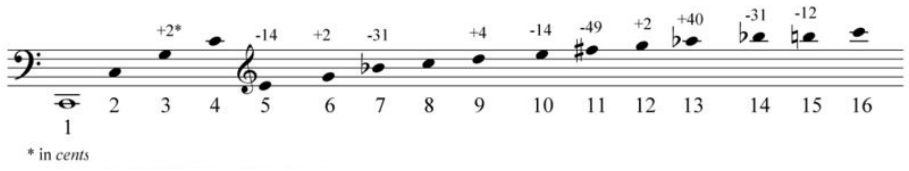


Figure 1.3: The series of harmonics.

This discovery gave an important physical justification to Rameau's theory and led him, in his *Gèneration harmonique ou traitè de musique thèorique et pratique* (1737), to the formulation of a new metaphysical concept: the *corps sonore*, final unification of the abstract theory of ratios and the coeval musical praxis. An example of this unification are geometrical progressions, that Rameau used to describe harmonic movements of the *fundamental*: these progressions were purely theoretical originally, but with the introduction of the *corps sonore* they became linked to musical praxis.

[2]The small numbers above the notes in figure 1.3 are the difference in *cents* between the natural frequency and the equal-tempered frequency. A cent is a logarithmic unit of measure used for musical intervals; in twelve-tone equal temperament each octave is divided into 1200 cents. More information about the equal temperament will be given in section 1.1.3.

Although Rameau's theory produced important results, they hadn't been accepted initially from the scientific community: he submitted his *Gèneration harmonique* to the Acadèmie Royale de Sciences but didn't get good reviews. Only in 1750 Rameau managed to have some recognition, thanks to some comments from D'Alembert: he wrote that harmony, with Rameau's research, gained a new rational status not subject anymore to arbitrary laws.

Ratio, *sensus* and the classification of intervals

Rameau's important theoretical formulations of the *fundamental* and of the *corps sonore* and his efforts to merge the mathematical theory of ratios with musical praxis is a good example of the eighteenth-century dicothomy between *ratio* and *sensus*.

A few years after Rameau's books, there were still theoreticians that proposed purely mathematical theories for the classification of intervals. Giuseppe Tartini, for example, in his *Trattato di musica secondo la vera scienza dell'armonia* (1754) described musical intervals and chords using the ratio between the circumference and its diameter. Leonard Euler (1707-1783), in his *Tentamen novae theoriae musicae* (1739), described a general algorithm to compute the consonance degree for any interval of the form $1/P$, where P is a positive integer number, decomposing it into its prime factors. The classification of intervals proposed by Euler, assigned a higher degree of consonance to intervals containing prime numbers: the octave $(2:1)$, that was unanimously considered the most consonant of all intervals, gained for this reason a low degree of consonance. This was in contrast to the coeaval musical praxis and his contemporary Johann Mattheson (1681-1764) criticized a lot Euler's approach, claiming that a musician doesn't need any kind of mathematical knowledge to evaluate and classify intervals.

The birth of *musical logic*

While Rameau, with his theory, established a clear connection between the rational interpretation of chords and their physical properties, his illuministic approach stressed the importance of a *logical* foundation for musical theory.

The application of *reason* to mathematics is prominent in the work of the theoretician Hugo Riemann and of his predecessors Hauptmann and Oettingen. They explored the logical relations present inside and between consonant chords: they probably didn't know Boole's work; nonetheless they proposed the idea of mathematical logic in music as an instrument to evaluate the *syntax* of the language.

Riemann's work is focused primarily on two phenomena: the harmonic dualism and the system of tonal relations . The harmonic series presented in figure 1.3 is very imporant in the definition of the major chord. However, it is not possible to define minor chords using this series. Riemann solved this problem by defining minor chords as the *dual* of major chords: the latter are generated by the harmonic series on the fundamental, while the former are generated by the harmonic series on the fifth harmonic of the latter. In other words, he organized a dualistic construct made of major and minor chords sharing the fundamental (for example: the note C in the structure F-Ab-**C**-E-G) thus merging the physics of chords with a logical structure.

Riemann's system of tonal relations illustrates the *modularity* and *finiteness* of the major/minor tonalities, organizing all tones in twelve pitch classes. This approach is the bases of all modern theories of the tonal pitch space and will be extensively illustrated in following chapters. Table 1.1 shows the system of tonal relations defined by Riemann.

Table 1.1: Riemann's system of tonal relations.

$e_\sharp/f$	c	g	d	a	e	b	$f_\sharp$	$c_\sharp$	$g_\sharp$	$d_\sharp$	$a_\sharp$
$c_\sharp$	$g_\sharp$	$d_\sharp$	$a_\sharp$	$e_\sharp$	$b_\sharp/c$	g	d	a	e	b	$f_\sharp$
a	e	b	$f_\sharp$	$c_\sharp$	$g_\sharp$	$d_\sharp$	$a_\sharp$	$e_\sharp/f$	c	g	d
f	c	g	d	a	e	b	$f_\sharp$	$c_\sharp$	$g_\sharp$	$d_\sharp$	$a_\sharp$
$d_\flat$	$a_\flat$	$e_\flat$	$b_\flat$	f	c	g	d	a	e	b	$f_\sharp$
a	$e/f_\flat$	$c_\flat$	$g_\flat$	$d_\flat$	$a_\flat$	$e_\flat$	b	f	c	g	d

The expression *musical logic* appears, for the first time, in the title of Riemann's first work *Musicalische Logik: Ein Beitrag zur Theorie der Musik*. In this book Riemann tries to define the application of logic to music as a device to

control the grammar of the harmonic syntax. He defines special symbols to represent movements on the system of tonal relations: Q for a fifth up and T for a major third down (and relative inversions as $-Q = \frac{1}{Q}$ and $-T = \frac{1}{T}$). He considered harmonic progressions as operations on the system that can be precisely measured as movement on the table 1.1. More information on this approach will be given in chapter 3.

1.1.3 The XX and XXI centuries: the compositional perspective

The efforts done by Riemann to logically organize the syntax of musical harmony had a great impact on the development of the domain, bringing the link between mathematics and music at a clear and established level.

Unitll XX century, however, this relation was mainly unidirectional: the music was *based* on mathematics and used it in order to organize its processes. In other words, the role of mathematics has always been secondary and it has been used only as *description* of musical phenomena.

At the beginning of XX century the situation changed: composers started using actively mathematical concepts to create new music and the link between the two domains became even stronger than before. Mathematics entered the inner mechanics of musical creation: it actually replaced traditional musical rules such as harmony, counterpoint and so on.

Serialism and pitch-class set theory

The intellectual movement that initially promoted this new usage of mathematics is called *serialism*. This movement has its roots in the european musical tradition and is directly connected to the work of Arnold Schönberg on *twelve-tone technique*; the main composers involved in its development were, among the others, Anton Webern, Pierre Boulez, Karlheinz Stockhausen and Luigi Nono. Around 1950, in the United States, a parallel movement exploited the usage of set-theory in music: its name is *pitch-class set theory* and has been mainly developed by Milton Babbitt, Allen Forte, Howard Hanson and many others[3]. The results produced by pitch-class set theory have been

[3]For a more detailed historical introduction to serialism and pitch-class set theory please see (Verdi 1998)

greatly influential in the context of contemporary musical theory and will be examined in a dedicated chapter.

The massive usage of mathematics for musical creation, raised problems in the estethic domain. Many musicians started thinking that the music created with such a system would have suffered of some form of mechanicism. It must be noted, however, that the dialectic between the objective generative procedures and their subjective musical application is probably the distinctive feature of contemporary musical theory and it's not an easy matter to decide how much matemathics affects the estethics of creation; for this reason, this problem will be not discussed further on.

Equal temperament and $\mathbb{Z}_{12}$

The new relation between mathematics and music in the modern era, is directly linked to the new way of representing intervals appeared around XVII century: the *equal temperament*. This is a new system of tuning in which every pair of adjacent elements has the same ratio. In other words, in equal temperament tunings the octave (whose ratio is $2:1$) is divided into a series of equal steps. For european classical music the number of steps is usually 12, each division being represented by $\sqrt[12]{2}$.

In equal temperament, any interval can be *reduced* to a quantity smaller then the octave through the modulo operator; moreover, any set of intervals can be *transformed* by different operators still preserving its properties. This lead to a completely new conception of the space of musical pitches, based on the properties of *ciclicity* and *simmetry*. Musical intervals and chords can finally be represented as *integers* in a mathematical structure called *group*, denoted by the symbol $\mathbb{Z}_{12}$. The structure has many interesting properties and all operations on intervals can be now defined formally as operations in a group.

As a matter of facts, the system of tonal relations presented by Riemann depicted in table 1.1 is also isomorphic (equivalent) to a group, exploiting the properties of *finiteness* and *inversion*. More on this isomorphism and on the group $\mathbb{Z}_{12}$ will be shown in chapter 3.

1.2 The need of symbolic representations

As appears from the above discussion, music and mathematics are intimately linked, both from physical and historical points of view. The key aspect of this link is *symbolic representation*: musical entities (such as intervals or chords) are represented in the formal language of mathematics with symbols. Through a symbolic language, the constitutive elements of musical syntax are organized in a logical way: the fruition of music is directly connected with the possibility of representing it in a symbolic way.

Symbolic representations have been used, initially, as a mean to describe the harmonic syntax; more recently, the formalism entered actively the creative process. The new usage of formalism exploited, principally, in two different but related ways coming from Riemann's theories: the *algebraic* approach and the *logical* one. The former has been mainly developed by the pitch-class set theory beginning from the fifties; the latter, has its roots in the application of formal logic to other fields happened during the seventies. Since both approaches massively affected musical creation, they will be extensively discussed in the following chapters.

Both approaches, however, share an important feature: they apply symbolic descriptions to object that are *already* symbolic. Music has its own language and special symbols: the notes on a score, the rhythm notation, the dynamics are complex entities that are described by ad-hoc symbols.

Music and musical signals

Music, takes place thanks to performance: the player decodes ad-hoc symbols and converts them into real *sounds*. There is, here, an essential separation: written music *is not* actual music; only in the final stage of performance music becomes physically real, when it become a *musical signal*[4]. The problem with both the algebraic approach and the logical one is that they only analyze music in the form of score: in other words, they symbolically represents objects that are already symbolic.

The main purpose of this research is to find a **new representation method**

[4]This expression involves many research fields and will be examined in details in chapter 4.

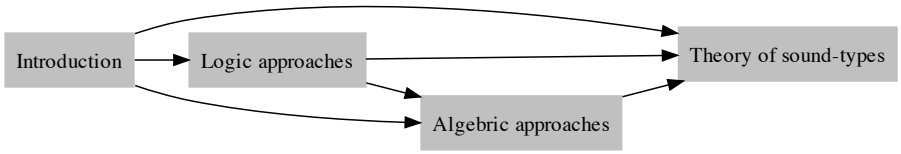


Figure 1.4: Outline of the chapters.

for music that takes into account both dimensions of music: the score (symbol) and the sound (signal). The reserach, moreover, tries to find answers to the following questions:

1. Which is the relationship between mathematical logic and musical logic?
2. Has the formalism based on *musical reasoning* something in common with logic formalism?
3. Can mathematical logic be useful to musicians, in order to clarify their reasoning?

The central idea of the approach proposed in this context is mixing the abstract level of symbolic logic with a computational level based on signal processing techniques and statistical analysis: it takes inspiration from types theory and particularly from the System F by the french logician J.Y. Girard.

1.3 Outline of the work

The organization of this work is the outlined in figure 1.4 and will be shortly described below.

Chapter 2 presents the results related to the so-called logical approach. The first attempts dates back to early XX century: they are due to Susanne Langer, an american philosopher that described a set of postulates for representing music using elementary logic; this approach has been expanded in the seventies by the logician Lennart Åqvuist. Other examined approaches are the one based on modal logic by Jos Kunst and the one based on functional calculus by Yann Orlarey.

Chapter 3 outlines all attempts for representing music throught algebraic structures: it starts from classification of chords and describes also the pitch-class set theory and the transformational theory by David Lewin. Some selected topic based on this approach are also presented.

Chapter 4 presents the novel contribution of this work, describing a method for representing music in a *quasi*-symbolic way using a mixture of symbolic and computational methods; this method is called the *theory of sound-types*.

Chapter 5, finally, tries to summarize the results obtained and discusses them in the perspective of musical creation: the final aim of the chapter is understanding if any of the presented approaches helps composers, performers, theoreticians and listeners to better understand and manipulate music.

A note on the used symbolism

This work connects different fields which have different formalisms. For this reason, it has not been possible to find a single style for the used symbolism.

In chapter 2 the formalism comes from logic and in particular from axiomatic set theory and λ-calculus. In chapter 3 the formalism has been taken from algebra and in particular from groups theory. In chapter 4, finally, the formalism is mainly related to digital signal processing and statistical data analysis. In the chapter, however, there is also the introduction of simple type theory that comes with its specific formalism.

While music has a natural symbolic representation in the *score*, this work deals principally with highly formalized representations related to the mathematical language; for this reason, no scores will be used to represent music.

Chapter 2

Logical approaches

Sometimes it seems as though each new step towards AI, rather than producing something which everyone agrees is real intelligence, merely reveals what real intelligence is not.

Douglas R. Hofstadter

Abstract

This chapter will present some attempts of representing music with logical formalism. The approaches being discussed are based, respectively, on the definition of postulates, on set-theoretic structures and on λ-calculus.

2.1 Susanne Langer's logic of music

In1929, the american journal *The Monist* published an article titled *A set of postulates for the logical structure of music* by Susanne K. Langer (Langer 1929). That article put new light on the problem of the foundation of music.

After the conceptual revolution operated by Schönberg and his school in early XX century, it was clear that a deep inquiry into the *metalevel* of music was needed and composers started thinking about *compositional theories*. One of the first and more formalized answers to this problem was given by Langer's logical approach.

2.1.1 Complexity and abstract form

Each universe of discourse has its logical structure and there is a finite number of possible situations that may happen in it. For simple universes, the

empirical discovery of situations could be affordable. There are cases, however, in which complexity is so high that an exhaustive approch is not possible. Arts, ethics and science in general are usually so complex that is almost useless to interpret them as purely logical situations. It's very difficult, for example, to study all their *possibilities* by means of logical deduction; the only solution is looking for some simple relations between basic elements.

A part of modern logic also focuses on the research of general fundamental properties that can be used as *postulates* to describe all situations of a universe of discourse. The reduction of a great number of situations to few postulates decreases the general complexity and creates an *abstract form* of the universe being represented. In most cases, of course, this possibilty remains purely theoretical because of the huge amount of material to be analysed. There are situations, however, in which the reduction to an abstract form is feasible. Susanne Langer supposes that music is such a situation.

2.1.2 The fifteen postulates

Langer supposes that there are relatively little elements involved in music and that there are only a few possibilities to combine them following definite principles. This set of postulates should represent the abstract form of music (the *logic of music*) and should be able to describe all possible musical situations. The abstract form of music is itself similar to a special algebra, neither numerical or Boolean, but of equally mathematical form and for which there exists at least one interpretation.

The following set of postulates aims to define such abstract form.

Let K be a class of elements $a, b, c, \ldots$, $\cdot$ and $\rightarrow$ binary operations, C a monadic relation and $<$ a dyadic relation. Then:

1. if $a, b \in K$ then $a \cdot b \in K$;
2. $\forall a \in K, a \cdot a = a$;
3. if $a, b \in K$ then $a \rightarrow b \in K$;
4. $\forall a, b \in K, a \rightarrow b = b \rightarrow a \implies a = b$;
5. $\forall a, b, c \in K, (a \cdot b) \cdot c = b \cdot (a \cdot c)$;

6. $\forall a, b, c \in K, \exists d \in K$ such that $(a \to b) \cdot (c \to d) = (a \cdot c) \to (b \cdot d)$;
7. $\exists r \in K$ such that $\forall a \in K, a \cdot r = a$;
8. there is a K-subclass T such that $\forall a, b \in K$ other than r and $\forall c \in K$, if $a = b \cdot c \implies b = c$ and $a = b \to c \implies b = r \vee c = r$ then $a \in T$;
9. $\forall a \in T, C(a \cdot a)$;
10. $\forall a, b, c \in K, \neg C(a \cdot b) \implies \neg C(a \cdot b \cdot c)$;
11. $\forall a \in K$ there exists K-subclass A such that $\forall b, c \in K, b \in A \iff C(a) \cdot b \equiv C(b) \cdot c$;
12. $\forall a, b \in T$ with $a \neq b$, $\neg(a < b) \equiv (b < a)$;
13. $\forall a, b, c \in T, a < b \wedge b < c \implies a < c$;
14. $\forall a, b \in T$ where $b \notin A$ and $\forall a\prime \in A$, $\exists b\prime \in B$ such that if $a < a\prime$ then $\neg(a < b < a\prime) \implies (a < b\prime < a\prime)$;
15. $\forall a \in T, \exists a^\circ \in A$ such that $\forall b \in A$ with $b \neq a, b \neq a^\circ$, $a < b \wedge a < a^\circ \implies a^\circ < b$ and $a < a^\circ \wedge b < a^\circ \implies b < a$.

Differences with Boolean algebras

The set of postulates described above is very similar to a Boolean algebra. The operations $\cdot$ and $\to$ are similar to common sum and multiplication. While $\cdot$ is commutative and associative like Boolean $\times$, however, $\to$ is not commutative and differs from Boolean $+$. Morover, while r corresponds to Boolean 0 there is not the equivalent to Boolean 1. The dyadic relation $<$ has similar properties to Boolean *inclusion* but $<$ is only applied to K-subclasses.

Summarizing, the major differences between the set of postulates described above and a Boolean algebra are:

- non commutativity of $\to$;
- incomple nature of the neutral element r;
- the lack of neutral element for $\cdot$ (corresponding to Boolean 1).

These divergences make the new algebra less symmetric than the logical calculus. Duality of $+$ and $\times$ is not preserved; moreover, the tidiness of the system is lost due to the incomplete nature of the neutral elements.

2.1.3 Interpretation of the new algebra

The interpretation of the given postulates leads to the definition of the *formal structure of music*. In order to properly understand this interpretation, it's needed to rethink musical elements in a more general sense. The logical approach proposed by Langer, in fact, deals not only with major or minor intervals (that are linked to a specific musical context) but is general enough to handle non tempered systems, harmonic theories not based on triads and so on. For this reason, no elements of any specific compositional theory or style have been included, in order to keep the interpretation general enough. The basic postulates can be interpretated as follows:

1. if a and b are musical elements, the *interval* $a \cdot b$ is a musical element;
2. if a is a musical element, it is equal to the *unison* $a \cdot a$;
3. if a and b are musical elements, the *progression* $a \rightarrow b$ is a musical element;
4. if a and b are musical elements and if $a \rightarrow b = b \rightarrow a$ than a and b are the same musical element;
5. if a, b and c are musical elements, then the interval $(a \cdot b) \cdot c = b \cdot (a \cdot c)$.
6. if a, b and c are musical elements, then exists the musical element d such that the interval of progressions $(a \rightarrow b) \cdot (c \rightarrow d)$ is equal to the progression of intervals $(a \cdot c) \rightarrow (b \cdot d)$[1];
7. there is at least one musical element r such that, if a is a musical element other than r, the interval $a \cdot r = a$;
8. there is a subclass T of musical elements (called *tones*) such that, if a and b are musical elements other than r, c is a musical element and if

[1] Langer suggests that this postulate embodies the principle of counterpoint.

$(a = b \cdot c) \implies (b = c)$ and $(a = b \to c) \implies b = r \vee c = r$, then a is a tone (in other words, if a is an interval it is a unison, and if a is a progression every memebr but one is a rest);

9. if a is a tone, the unison $a \cdot a$ is *consonant*;

10. if a, b and c are musical elements and $a \cdot b \cdot c$ is consonant then $a \cdot b$ is consonant;

11. for any musical element a other than r, there is a subclass A (called *recurrences*) such that for any b and c, b is the recurrence of a if and only if $(a \cdot c$ is consonant$)$ is equivalent to $(b \cdot c$ is consonant$)$;

12. if a and b are two different tones and a is not before b in order of pitch, then b is before a;

13. if a, b and c are tones, then if a is before b and b is before c then a is before c;

14. if a and b are distinct tones with b not being recurrence of a and $a\prime$ is a recurrence of a, then there is at least one $b\prime$, recurrence of b, such that if a is before $a\prime$ and b is not between a and $a\prime$, then $b\prime$ is between a and $a\prime$ in order of pitch;

15. For any tone a, there is a tone a°, recurrence of a, such that if b is any other recurrence of a and if a is before b and also a is before a° then a° is before b; and if a is before a° and b is before a°, then b is before a in order of pitch (in other words, there is at least a recurrence of a called *octave* such that no other recurrence can lie between it and a).

2.1.4 Fundamental theorems

The essential relations between musical elements, such as the repetitional character of the order of tones within the octave or the equivalence of consonance-values of any interval and any repetition of iteslf, can be deducted from the given postulates. The following is a list of the most important theorems that can be deducted from the postulates; they are given without proof, for more information see (Langer 1929).

1. THEOREM. *For any* $a, b \in K$*:* $a \cdot b = b \cdot a$*.*

2. THEOREM. *For any* $a, b, c \in K$*:* $(a \cdot b) \cdot c = a \cdot (b \cdot c)$*.*

3. THEOREM. *For any* $a, b \in T$*, any* $a\prime \in A$ *and any* $b\prime \in B$*:* $C(a) \cdot b \equiv C(a\prime) \cdot b\prime$*.*

4. THEOREM. *For any* $a \in T$ *and any* $a\prime \in A$*:* $C(a) \cdot a\prime$*.*

5. THEOREM. *For any* $a, b \in T$*:* $b \in A \implies a \in B$*.*

6. THEOREM. *For any* $a, a^\circ, b \in T$ *and any* $b\prime \in B$*:* $(a < b < a^\circ) \cdot (a < b\prime) \implies a^\circ < b\prime$*.*

7. THEOREM. *For any* $a, b, c, a^\circ, b^\circ \in T$*:* $\exists c^\circ \in C$ *such that* $(a < b < a^\circ < b^\circ) \cdot (a < c < b) \implies (a^\circ < c^\circ < b^\circ)$*.*

2.1.5 Generalization to particular musical systems

There are, probably, many other relations between musical elements that can be derived from the given postulates. Nonetheless, even a complete development of the theory would only lead to the formalization of *general* musical possibilities. More specifications are needed, for example, to deal with occidental classical music:

- the *next-member* for the series generated by $<$;
- the determination of consonant intervals other than the unison and the octave (and eventually also a way to *order* consonances);
- the introduction of the specific T-functions $\sharp$ and $\flat$.

It is possible to formalize different music systems by imposing special constraints on K. For example, to describe Hawaiian music it would be needed a postulate for continuous-series on pitch; in Gaelic music, the concept of consonance should be redefined since adiacent tones do not produce dissonanaces; and so on.

At the end of her paper, Langer wonders if it could be possible to apply this approach also to other arts in order to find a common background for

comparisons. Philosophycal implications of such a background would be enormous: psychology and metaphysics have failed to put aestethics on any better basis than the empirical one. Could logic fill the gap? Langer's paper ends leaving this question open.

While Susanne Langer proposed the described approach in an historical period focused on compositional theories, her paper did not receive so much attention. Only after about fifty years other researchers used her ideas to develop new tools for defining a logic of music; next sections will outline such attempts.

2.2 A set-theoretical point of view

The method proposed of by Susanne Langer is mainly focused on the harmonic paramter and does not pay attention to *temporal* evolution of musical elements. In order to add time to the formalization, it is possible to describe music from a set-theoretical point of view, by defining specific algebraic structures. This section will propose an approach based on the work done in the late twenties by Susanne Langer and expanded in the seventies by Lennart Åqvuist; see (Langer 1929) and (Aqvist 1979).

2.2.1 Abstract musical systems

With *time-limited frame* is defined an ordered quadruple $\langle T, t-, -t, \leq \rangle$ such that $\forall(t, t', t'') \in T$:

T1. $T \neq \oslash$
T2. $t-, -t \in T$
T3. $t- \neq -t$
T4. $\leq \in T \times T$ (linear ordering)
T5. $t- \leq t$
T6. $t \leq -t$
T7. $t \leq t$ (reflexivity)
T8. if $t \leq t'$ and $t' \leq t''$ then $t \leq t''$ (transitivity)

T9. if $t \leq t'$ and $t' \leq t$ then $t = t'$ (antisymmetry)
T10. $t \leq t'$ or $t' \leq t$.

Similarly, a *frequency-limited frame* is an ordered quintuple $\langle P, p-, -p, \S, \leq \rangle$ such that $\forall (p, p', p'') \in P$ hold axioms **P1 - P10** defined in the same way as axioms **T1 - T10** and also holds:

P11. $\S \notin P$ (namely, the *null*-frequency).

Then, a *musical frame* is the structure $\langle \langle T, t-, -t, \leq \rangle, \langle P, p-, -p, \S, \leq \rangle, V \rangle$ where:

(i). $\langle T, t-, -t, \leq \rangle$ is a time-limited frame;
(ii). $\langle P, p-, -p, \S, \leq \rangle$ is a frequency-limited frame;
(iii). $V \neq \oslash$ (namely, the set of *voices*).

A *musical frame with voice-indexed temporal partitions* is the structure $F = \langle \langle T, t-, -t, \leq \rangle, \langle P, p-, -p, \S, \leq \rangle, V, S_v \rangle$ such as:

(i). $\langle \langle T, t-, -t, \leq \rangle, \langle P, p-, -p, \S, \leq \rangle, V \rangle$ is a musical frame;
(ii). S_v is a function from V to the power-set of T (namely, the *point-selector*) such as $\forall v \in V$, S_v is a finite subset of T and $t-$, $-t$ exist both in S_v.

With S_v it is possible to define the notions of *temporal segments* and *time interval* $\phi(S_v)$: this second concept is really important in the definition of temporally quantified systems given in paragrah 2.2.2.

On structure F, defined above, it is possible to define the *melodic-rhythmic specification* as the ordered couple of functions $\langle On, Att \rangle$ on V such as $\forall v \in V$, holds:

On_v and $Att_v \subseteq T \times (P \cup \{\S\})$.

Intuitively, On_v is a binary relation that associates *time-points* to *frequencies* specifying if a given frequency *sounds* at a given time in a given voice; in other words: $\forall t \in T, \forall p \in P \cup \{\S\}$, $\langle t, p \rangle \in On_v$ means that frequency p

sounds in the voice v at time t. In the same way, Att_v is a binary relation that specifies when a frequency *starts* in a given voice.

It is now possibile to define the *abstract musical system* as the structure $M = \langle F, \langle On_v, Att_v \rangle \rangle$, such as:

(i). F is a musical frame with voice-indexed temporal partitions;

(ii). $\langle On_v, Att_v \rangle$ is a melodic-rhytmic specification on F.

Then, the *stream of musical events of voice* $v \in V$ *in* M is the relation $ON_v \subseteq T \times (P \cup \{\S\})$ that, intuitively, *selects* from T all the *time-points* in which voice v *sounds*, starting from a given time. Similarly, the *chord at time* $t \in T$ *in* M is the relation $CH_v \subseteq T \times (P \cup \{\S\})$ that selects all the positive frequencies that *sound* in any of the given voices of M at the given time t.

Finally, it is possible to define the *texture of* M as the set of all the streams of musical events on M, i.e. $Texture(M) = \{ON_v : v \in V\}$ and also the *chords progression of* M as the set of all chords on M[2].

From a musical point of view, the *texture* represents the *counterpoint* of a musical composition, while the *chord progression* represent the *harmony*.

2.2.2 Temporally quantified abstract musical systems

Let F = $\langle \langle T, t-, -t, \leq \rangle, \langle P, p-, -p, \S, \leq \rangle, V, S \rangle$ be any musical frame with voice-indexed temporal partitions and $TI = \bigcup_{v \in V^{\phi(S_v)}} \cup \phi(\bigcup_{v \in V^{S_v}})$ the set of time-intervals determined by V and S.

With *measurement of duration* on F it's defined the ordered quadruple $\langle E, L, E^+, m \rangle$, where:

i E and L are the binary relations *is of the same length* and *is shorter than*, respectively in TI;

ii E^+ is a ternary relation on TI such that for any $x, y, z \in TI$: $< x, y, z > \in E^+ \iff E(x, y + z) = true$, ie. x *is of the same length* of y and z taken together;

[2]Formal definitions of all the concepts presented are beyond the scope of this chapter and can be found in (Aqvist 1979).

iii m is an *additive measure* for the structure $\langle TI, E, L, E^+\rangle$ in the sense of being function of TI onto some subset of the non-negative real numbers such that for all $x, y, z \in TI$ hold:

- $m(x) = m(y) \iff < x, y > \in E$;
- $m(x) < m(y) \iff < x, y > \in L$;
- $m(x) = m(y) + m(z) \iff < x, y, z > \in E^+$.

The relations defined above must be valid not only on the union of all interval-sets $\phi(S_v)$ for $v \in V$ but also on the result of adding to that union every member of the interval-set $\phi(\bigcup_{v\in V} S_v)$.

This condition is weak but necessary to be able to adequately measure the length of the time-intervals given by V and $S\pounds$. Moreover, to have a complete notion of measure of duration in F it's also necessary that E, L and E^+ satisfy some minimal conditions for the existance of a measure in the structure $\langle TI, E, L, E^+\rangle$. For instance, E must have an equivalence relation on TI and L must be a strict partial order in that set.

To that conditions must be added one for which the *null-stretch* $-t$ has to be such that $\langle \vec{-t}, \vec{-t}, \vec{-t}\rangle$ whence by virtue of the condition (iii)) above $m(\vec{t}) = 0$.

Also, $\vec{-t}$ must be the *only* member of TI that satisfies this condition; this condition is really important for the following situation. It is theoretically possible to use T as a finite set of *discrete* moments; in this case, the condition above imposes a strong restriction on the choice of the point-selector S: its cardinality must be at least

Finally, with *temporally quantified abstract musical sysyem* is defined the strucure $M_t = \langle F, \langle On, Att\rangle, \langle E, L, E^+, m\rangle\rangle$ such that:

- $\langle F, \langle On, Att\rangle\rangle$ is an abstract musical system;
- $\langle E, L, E^+, m\rangle$ is the measurement of duration of F.

Given any pair of abstract musical systems, either temporally quantified or not, the conditions for which they are *identical* are the standard ones from ordinary set theory. Nonetheless, many problems arise in *practice* when two

system must be compared: the reason of these problem is in the *actual* nature of music that appears after performance. More on this problem will be discussed in chapter 4.

Other important musical concepts can be defined by expanding the presented formalism, but this should be enough to sketch out main possibilities of the approach.

2.2.3 Perfect instantiation of abstract musical systems

It is worth, at this point, to notice an important fact about the formal theory presented above. So far, it has been assumed that a musical composition *can be represented* by an abstract musical system[3]. This is of course a very strong assumption and outlines one of the major problems of symbolic-level representations: a formal system, must have a *connection* to reality to be somehow useful; this connection is usually called *model* of the system. In a very general sense, a model is a set of relations that map each element of a theory to some truth values[4]. Any formal system needs to be *instantiated* in a possible situations over a *world* of possible situation to be effectively used as means of representation. An instance of the system described above could be, for example, a particular *performance* (of some music being analysed) over a world of existing performances; the instantiation process can be also formalized. Let $M = \langle F, \langle On, Att \rangle\rangle$ be an abstract musical system and $w \in W$ a possible instantiation of M over a world of instantiations W. With $PerfConc_w(M)$ (*perfect concretization of M in w)*, it's defined the structure $M^w = \langle\langle\langle T^w, t{-}^w, {-}t^w, \leq^w\rangle, \langle P^w, p{-}^w, {-}p^w, \S^w \leq^w\rangle,$
$V^w, S^w\rangle, \langle On^w, Att^w\rangle\rangle$, such as:

(i). $T^w = T \times \{w\}$;
(ii). $t{-}^w = \langle t-, w\rangle$ and ${-}t^w = \langle -t, w\rangle$ respectively;
(iii). $P^w = P \times \{w\}$;
(iv). $p{-}^w = \langle p-, w\rangle$, ${-}p^w = \langle -p, w\rangle$ and $\S^w = \langle \S, w\rangle$ respectively;

[3] In this context, a musical composition could even be a written *score* and doesn't need to be in the state of *performance*.

[4] This definition is of course very imprecise; in this context, anyway, it is only needed to present main ideas about *model theories*.

(v). $V^w = V \times \{w\}$;

(vi). $\leq^w$ is a binary relation on $T \times \{w\}$ such as $\forall (t, t') \in T : \langle t, w\rangle \leq^w \langle t', w\rangle \iff t \leq t'$;

(vii). $\leq^w$ is a binary relation on $P \times \{w\}$ such as $\forall (p, p') \in P : \langle p, w\rangle \leq^w \langle p', w\rangle \iff p \leq p'$;

(viii). S^w is a function from $V \times \{w\}$ to the power-set of $T \times \{w\}$, such as $\forall v \in V, \forall t \in T : \langle t, w\rangle \in S^w_{\langle v,w\rangle} \iff t \in S_v$;

(ix). $\langle On_w, Att_w\rangle$ is an ordered couple of functions on $V \times \{w\}$ such as $\forall v \in V, t \in T, p \in (P \cup \{\S\}) : \langle\langle t, w\rangle, \langle p, w\rangle\rangle \in On^w_{\langle v,w\rangle} \iff \langle t, p\rangle \in On_v$ and $\langle\langle t, w\rangle, \langle p, w\rangle\rangle \in Att^w_{\langle v,w\rangle} \iff \langle t, p\rangle \in Att_v$.

The *perfect concretization* determines an isomorphism for each $v \in V$, by projecting M onto M_w while preserving all releations and functions.

With *musical model*, finally, it's defined ordered triple $\mu = \langle \aleph, W, inst\rangle$ such as:

(i). $\aleph$ is a set of abstract musical systems;

(ii). W is a non-empty set (world) of *possible instantiations*;

(iii). $inst \subseteq \{PerfConc_w(M) : w \in W, M \in \aleph\}$ (i.e. is a selected subset of all perfect concretizations of M in w with $\forall w \in W$, while all $M \in \aleph$ are the selections of the perfect concretizations *really exhibited* in $w \in W$).

Roughly, all this formalism *means* the following:

- for each realistic $inst$ set, $PerfConc_w(M) \notin inst$, since there are no perfect concretizations of M in any *real* performance; however, W could contain *some* situations w' such as $PerfConc_{w'}(M) \in inst$;
- it should be possible to define an abstract system $M' \in \aleph$ that *formalizes* a particular instance w, since $PerfConc$ is an isomorphism and it is invertible; this means that it should be possibile to *recover* the underlying abstract system M';
- the outlined representation of musical systems is not based only on *scores* but also on some *concretizations* of them (namely, the *performances*).

The last point is important for our purposes: the presented approach can have enough expressivity to describe music both in its *static* and *dynamic* states (i.e. the scores and the performances). However, to be useful, it always needs a verifiable semantics: to *validate* the expressions in the theory it is mandatory to provide a *model* that maps to truth values. In the proposed formalization, the problem of providing such mapping is still open.

Moreover, there is a strong *knowledge imposition* from the theory: it modelizes music in terms of *harmony* and *counterpoint*, for example, even if these concepts could be completely irrelevant to the particular music being modeled. In other words, the supposed ontology of the theory is fixed *a-priori* and is transferred to the analysed domain.

In the following section, a different approach to music description based on *modal* logic will be presented.

2.3 A modal approach

In the late seventies, Jos Kunst proposed a theory to model the *dynamics* of music (Kunst 1976). His starting point was an approach based on modalities, with operators like *necessary* ($\Box$) and *possible* ($\diamond$), applied to propositions about musical concepts over time. The propositions can refer to any kind of heard *sonic object* and the modalities help to change the modelization over time. For example, proposition $\Box c$ stands for *c is true always in past and present*, while $\diamond c$ stands for *c is true at some time in past or present*; these propositions can always be stated whatever is the concept represented by c. While listening to the musical flow, an object can change its modality passing from, say, $\Box c$ to $\diamond\Box c$: this change is also embedded in the proposed formalism through a special non-monotonic function called *bi-valence (BivFunc)*. The main purpose of the bi-valence function is to formally describe the process of *learning* and *unlearning* concepts. The expression $\Box c$ may be valid at time t_0 but not at time t_1: then, due to $-c$, the new expression $-\Box c \wedge \diamond\Box c$ is derived. The *learning* part of the bi-valence function introduces a new concept with its modality and produces a new law, say $\Box d$. Subsequently, c is unlearned by means of a different time line from the one used for conceptualization of d. The bi-valence function $BivFunc(c, d)$ then, crosses both

time lines and expresses the change of conceptualizations as an unlearning and learning process. Figure 2.1 represents a typical Kunst's diagram for the following expression:

$$BivFunc(c \rightarrow \neg e, e \rightarrow b, (\neg b \wedge e) \rightarrow a \wedge d)).$$

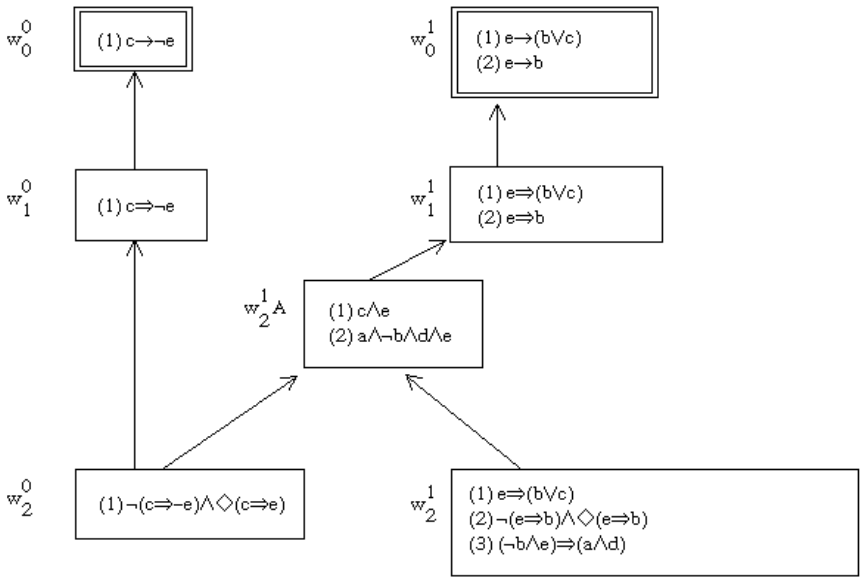


Figure 2.1: A typical Kunst's diagram for his $BivFunc$.

Musical applications

In a general sense, this theory is a sort of propositional calculus with temporal quantifiers; this gives to the theory an enormous expressive power: formal propositions could stand for *we hear section B of the composition*, for *the sound of the violin* or *the pitch D is not sounding any more* and so on. Such kind of expressions can be, of course, extremely useful to manipulate, in a very compact way, concepts that can be difficult to define in other ways: many composers, for example, admit to use metaphorical descriptions in

some phases of their activity. On the other hand, between the expression $\Box c$, meaning *the sound of the violin*, and the real sound of the violin there are no evident relationships. We return to the original problem of finding a *model* to the theory: there is no evident criterion to map the theorems to truth values; the only thing the theory can express are formal relations between *empty symbols* through logical operators.

2.3.1 Models by musical images

Symbolic-level representations can be expressive enough to describe complex relationships and hierarchies between concepts but are hardly related to the physical nature of sound and are usually non-invertible: they are based on *logical rules* that cannot be easily verified by any model.

A possible solution to provide a model to formal theories of sound has been proposed in (Leman 2002) in the context of logical formalization of musical coherence and is based on *perceptual* models.

The basic idea is to build an interpretation of expressions involving musical coherence through computational descriptions of musical content; this is achieved by defining the notion of *musical image*, a spatio-temporal representational entity that provides a link between sounds and their conceptualizations.

The creation of a musical image is the result of low-level features extraction processes applied to sound signals (see section 4.4.1). For this reason musical images formally represent properties closely related to sound. In order to create a perceptual model, musical images must be processed through the so-called *auditory* system; in terms of signal processing this corresponds to the application of operations such as filtering, correlation and so on. Musical images can be created at different time scales, since they relate to different aspects of perception and can involve memory. Moreover, each image can have a different degree of abstracion from the signal itself. Once images are defined, it is possible to *transform* them by means of special functional operators: this leads to a versatile model in which modal and temporal formalizations are possible.

A simple example will clarify the process. Let $c \rightarrow n$ be a valid expression of a formal propositional language; if c means *sound of the clarinet* and

n means *nasal timbre* then we can say that the whole expression states that the sound of the clarinet *implies* the nasal timbre. As specified in section 2.3 the expression is a simple relation between empty logical symbols; however, with musical images and their transformations it is possible to provide a model to that expression. It is possible to, for example, the image of *sound of the clarinet* and the image of *nasal timbre* as vectors $\overrightarrow{A}$ and $\overrightarrow{B}$ of low-level features computed from real signals and filtered through the auditory system. If, after a computational stage, it's discovered that it is possible to transform vector $\overrightarrow{A}$ into vector $\overrightarrow{B}$ then the formal expression $c \rightarrow n$ is said to be *valid* for that model.

This method is really promising: providing formal models by means of low-level features is an extremely strong idea. In the described approach the underlying logic assumes statically predefined concepts that are *imposed* onto the described sound: the concept of nasal timbre *must* be explicitly defined and the corresponding musical image *must* be deliberately created from the signal. Unfortunately, this clashes with the requirements for music representations defined in section 4.1: the semantics of the underlying logic should be signal-dependent in order not to impose predefined concepts.

2.4 A functional approach

Logical formalization showed in previous sections were mainly based on abstract formal logic. The developments of computers in the last century, created a new approach to logic based on computation: programming languages.

Basically, any programming language descends from a theoretical language called λ-calculus, proposed by Alonso Church (Church 1985). With programming languages, many problems have been formalized in a *practical* way and have been successfully solved; also music has been formalized through this approach.

In 1994, Orlarey, Fober, Letz and Bilton (Orlarey et al. 1994) proposed a new model to apply programming languages to music: instaed of building music data structures and functions on existing languages, they suggested to build suitable programming languages on music data structures. The pro-

posed approach is based on two main steps:

- extend a descriptive language by means of λ-abstraction and λ-application;
- extend the λ-calculus reduction rules in order to deal with specific language constructions.

In order to illustrate their approach, they initially created a graphical calculus that deals with 3D objects; later on, they transferred this model to music.

2.4.1 A graphic calculus

The following syntax defines a descriptive language that handles coloured cubes and relative operators:

$$
\begin{aligned}
cube ::= & \; color \\
& \mid [cube_1 | cube_2] \\
& \mid \left[\frac{cube_1}{cube_2}\right] \\
& \mid [cube_1 \; / \; cube_2] \\
color ::= & \; white \mid red \mid green \mid invisible \mid \ldots
\end{aligned}
$$

The meaning of the rules of the language is depicted in figure 2.2: this kind of interpretation is very important when dealing with practical applications. An more complex example is the following:

$$\left[\frac{white|invisible}{invisible/white} / \frac{invisible|green}{green|invisible}\right]$$

whose meaning is depicted in figure 2.3.

While the described language is powerful enough to create coloured cubes, it is not *generative*. In other word, it is not possible to create objects *other* than the ones defined in the rules of the language. In order to add expressivity to the language and to transform it from a descriptive-only to a real programming language, the following extensions should be added:

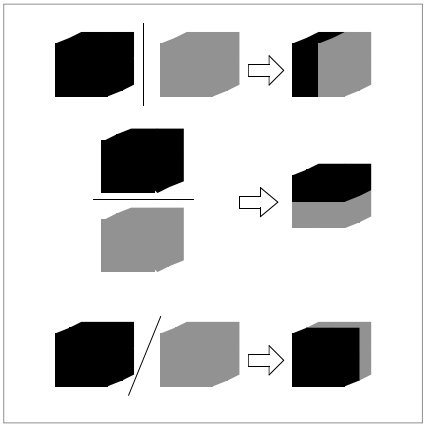


Figure 2.2: The meaning of the descriptive graphic language.

Figure 2.3: An extended example of the graphic language.

- *abstraction* and *application* rules;
- reduction rules to deal with the application of colored cubes.

This modifies the grammar of the language and create and extended syntax, described below:

$$
\begin{aligned}
cube ::= & \; color \\
& \mid [cube_1|cube_2] \\
& \mid \left[\frac{cube_1}{cube_2}\right] \\
& \mid [cube_1/cube_2] \\
& \mid \lambda color.cube \; (\textbf{abstraction}) \\
& \mid (cube_1 cube_2) \; (\textbf{application}) \\
color ::= & \; white \mid red \mid green \mid invisible \mid \ldots
\end{aligned}
$$

A β-reduction example

Applying the new rules to expressions of the language, it is possible to create *variables* that handle colors. The example given above becomes then:

$$
\begin{aligned}
\lambda white.\lambda green.\left[\frac{white|invisible}{invisible/white}/\frac{invisible|green}{green|invisible}\right] \; \textbf{blue} \; \textbf{red} \Rightarrow_\beta \\
\lambda green.\left[\frac{blue|invisible}{invisible/blue}/\frac{invisible|green}{green|invisible}\right] \; \textbf{red} \Rightarrow_\beta \\
\left[\frac{blue|invisible}{invisible/blue}/\frac{invisible|red}{green|red}\right]
\end{aligned}
$$

where *white* and *green* are now variables.

Construction of a diagonally divided cube

It is now possible to use recursion properties of the new language by defining an object as the application on itself; this leads to the creation of new objects. The following objects A and B will help to undestand this process:

$$
A = \left[\frac{green|red}{red|white}\right] \quad B = \left[\frac{green|\frac{green|red}{red|white}}{\frac{green|red}{red|white}|white}\right].
$$

They are depicted in figure 2.4 (a): it is obvious that in B the object A is repeated twice. In other words, it is possible to find a *general* expression that handles both objects:

$$X = \left[\frac{green|X}{X|white}\right]$$

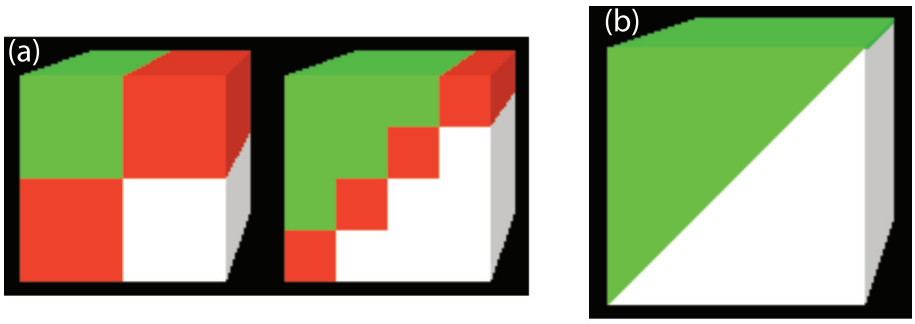


Figure 2.4: Figure (a) represents the repetition of cube A into cube B; figure (b) represents a diagonally divided cube.

This opens the possibility of creating objects that are *not* described in the rules in the language, such as the diagonally divided cube depicted in figure 2.4 (b); by applying the rule sketched above, the corresponding program will be:

$$X = \left(\lambda red.\left[\frac{green|(red\ red)}{(red\ red)|white}\right]\ \lambda red.\left[\frac{green|(red\ red)}{(red\ red)|white}\right]\right)$$

2.4.2 A music calculus

The application of the presented ideas to musical description can be done by redefining the *atomic* element of its grammar. Instead of colors and cubes, the new language will deal with pitches, octaves and so on. The grammar of this new calculus is defined below:

$$
\begin{aligned}
score &::= \phi \mid event \mid [score_1; score_2] \\
&\mid \left[\frac{score_1}{score_2}\right] \\
&\mid \lambda event.score \mid (score_1\ score_2) \\
event &::= r \mid note \mid event\ tmodifier \\
note &::= pitch \mid pitch\ octave \mid note\ nmodifier \\
pitch &::= c \mid d \mid e \mid f \mid g \mid a \mid b \\
octave &::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\
tmodifier &::= . \mid * \mid t \mid / \\
nmodifier &::= + \mid - \mid > \mid <
\end{aligned}
$$

Some β-reductions for music

The following example show the expression for a *3 time repetition*:

$$
\left(\lambda c.[c;c;c]\left[\frac{\frac{c4}{e4}}{\frac{g4}{b5}}\right]\right) \implies_{\beta} \left[\left[\frac{\frac{c4}{e4}}{\frac{g4}{b5}}\right]; \left[\frac{\frac{c4}{e4}}{\frac{g4}{b5}}\right]; \left[\frac{\frac{c4}{e4}}{\frac{g4}{b5}}\right]\right];
$$

A *canon form* can be expressed, instead, with the following formula:

$$
\left(\lambda c.\left[\frac{}{[r;c]}\right][c4;e4;g4;c4]\right) \implies_{\beta} \left[\frac{\text{[c4;e4;g4;c3]}}{\text{[c4;e4;g4;c3]}}\right]. \tag{2.1}
$$

The real power of this approach, however, appears when using recursion to generate *infinite sequences*:

$$
X \equiv (\lambda f.[c4;(ff)]\lambda f.[c4;(ff)])
$$

whose meaning is depicted in figure 2.5; these kind of expressions are really convenient for some musical activities that are related to repetition.

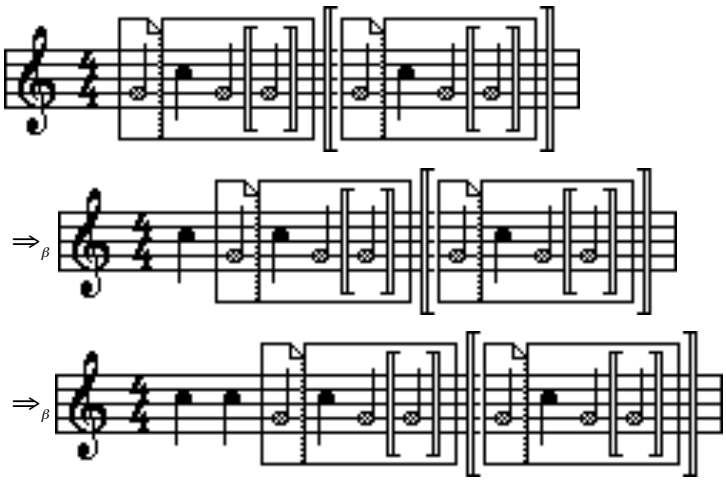


Figure 2.5: An infinite sequence using the described music calculus

Elody: a real implementation

Real implementations of functional music calculi exist: among the most important there are, for example, *Elody* (Orlarey et al. 1994) and *Haskore* . The following are a tipical sentences in Elody, expressing equivalences:

```
C/4+7 = C+7/4 = G*1/4
[C,_,E,F]/8 = [C/8,_/8,E/8,F/8]
{C,E,G}+2 = {D,F#,A}
Lambda x:M.{x,x+4,x+7,x+11}*2 = Lambda x:M * 2.{x,x+4,x+7,x+11}
Lambda x:M.{x,x+4,x+7,x+11} N = {N,N+4,N+7,N+11}
```

The Elody programming language is essentially a music language extented with λ-calculus; it clearly shows that the functional programming model is of great interest for music languages. A real interesting feature of this approach is that, being musical objects, functions can be represented as real music objects. In this way, programming becomes naturally linked to musical composition.

2.5 Summary

This chapter showed some possible approaches to represent and manipulate musical structures with formal logic.

The first approach discussed dates back to 1929 and is due to Susanne Langer. She defined a set of fifteen postulates with which represent most basic musical elements such as intervals, progressions and tones. Unfortunately, her ideas are very general and should be specialized with the addition of dedicated operators in order to be really useful for a particular musical system.

Langer's ideas have been expanded in the second approach discussed, due to Åqvuist. He focused on the concept of abstract musical system and defined all its properties. This approach is more powerful of Langer's because it also add time to the represented object, becoming more suitable for musical purposes.

Finally, a more practical approach has beed presented. It is based on functional languages, namely on λ-calculus. Through λ-application and λ-abstraction is possible to create generative languages that are able to perform recursion. A real implementation has been also showed, Elody, with some examples related to music.

Chapter 3

Algebraic approaches

How I need a drink, alcoholic of course, after the heavy chapters involving quantum mechanics.

György Pólya, using the *pilish* language

Abstract

3.1 Introduction

Chapter 1 outlined the relation between music and mathematics from an historical standpoint. This relation has been interpreted, traditionally, as the *application* of mathematical methods to music in order to describe phenomena. Since XX century, however, this perspective changed in favour of a new dynamic interaction creating a new research field called *mathemusic*.

This new world brought developments in both the original fields: musical problems can be formalized into mathematical statements; then, through the process of generalization, they became theorems. It's finally possible to apply theorems to describe new musical theories. Figure 3.1 depicts the discussed concepts.

Chapter 2 showed how the process of formalization has been done using logical tools; another possible approach to perform this formalization is through algebraic methods. The main fields of application for algebraic methods have been, historically, theoretical aspects, musical analysis and musical composition. In the recent developments the processes of analysing and writing music became very close and influenced each other.

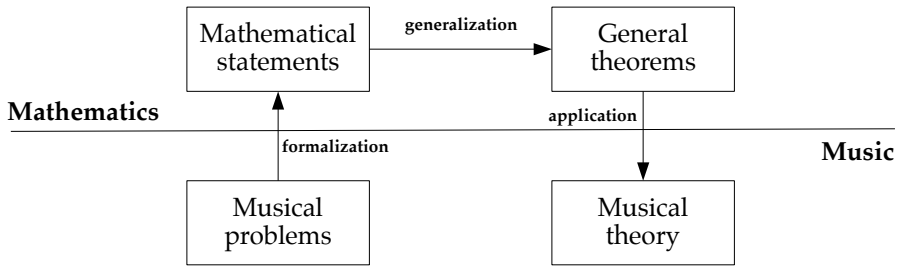


Figure 3.1: The dynamic interaction between mathematics and music.

Among the composers/theoreticians that mainly influenced this approach, there are Milton Babbitt (Babbit 1960) , Allen Forte (Forte 1977) , George Perle (Perle 1991) , David Lewin (Lewin 2007) and, more recently, Henry Klumpenhouwer (Jedrzejewski 2006) . They produced important results for the main mathemusical theories: the *pitch class set theory* and the *transformational theory*. For an historical introduction see (Verdi 1998).

Almost all the approaches based on algebraic methods rely on the possibilty of representing music thorugh the *group* mathematical structure; this possibility comes from the organization of the pitch with tempered-tuning systems (described in paragraph 1.1.3) and is directly connected to the approach proposed by Riemann[1]. In particular, the used groups are the cyclic, the dihedral, the affine and the symmetric one.

The following is a (partial) list of problems and theories that have been studied through the algebraic approach:

- classification of chords;
- set theory applied to pitches (*pitch class set theory)*;
- generalization of set theory by transformations (*transformational theory)*;
- musical mosaicing (for rhythms and pitches);
- Z-relation and homometric sets.

[1] All these theories are often called neo-rimannian approaches.

In the following sections, only some of them will be dicussed. After a short recall of the involved algebraic structures, it will be given a technical overview of the problem of counting chords, of the pitch class set theory and of the transformational theory. The problem of musical mosaicing will only be partially sketched out at the end of the chapter through a specific example regarding hexachords and their trichordal generators.

3.2 Algebraic background

The following section will summarize the required algebraic background to formulate Pólya's enumeration theory, needed to solve the problem of counting chords. Basic algebraic knowledge is required, in particular regarding groups theory.

3.2.1 Group actions

It's called *group action* of a multiplicative group G on a set X the mapping $G \times X$ on X, $(g, x) \rightarrow g.x$ that satisfies:

1. $1.x = x \ \forall x \in X$, where 1 is the unit element of G;
2. $(gh).x = g(h, x) \ \forall g, h \in G$ and $x \in X$.

Group actions induce the equivalence relation $x \sim y$ given by $\exists g \in G, y = g.x$, whose equivalence classes $G(x)$ are called the *orbits* of G on X and are defined as $G(x) = \{g.x | g \in G\}$; two elements $x, y \in X$ are said to be in the same orbit if $\exists g \in G$ such that $y = g.x$. The set of all possible orbits is denoted by $G\backslash\backslash X = \{G(x) | x \in X\}$. An action is said to be *transitive* if there exists only a single orbit.

The set G_x is called *stabilizer* for $x \in X$ and is defined as

$$G_x = \{g \in G | g.x = x\}. \tag{3.1}$$

The set X_g for $g \in G$ is called *fixed points* and is defined as

$$X_g = \{x \in X | g.x = x\}. \tag{3.2}$$

The following results about subgroup properties are necessary to understand Pólya's theorem discussed in next section and will be given without proofs; for more information see (Jedrzejewski 2006).

1. LEMMA. *If H is a subgroup of G, then the set G/H of the right cosets and the set $H\backslash G$ of the left cosets have the same cardinality: $|G/H| = |H\backslash G|$.*

It's called *index* of the subgroup H of G the cardinal of the left/right cosets defined as

$$|(G : H)| = |G/H| = |H\backslash G|. \tag{3.3}$$

The index of the trivial subgroup $H = \{1\}$ is called *order* of G and is denotated by $|G|$. A subgroup is defined *normal* if the right cosets are equal to the left cosets:

$$\forall g \in G, \;\; gH = Hg. \tag{3.4}$$

8. THEOREM (LAGRANGE). *The order and the index of a subgroup H of a finite group G are both divisors of the order of the group:*

$$|(G : H)| = |G|/|H|. \tag{3.5}$$

9. THEOREM. *The order of the orbit of x is equal to the index of the stabilizer of one of its elements:*

$$|(G(x)| = |G : G_x|. \tag{3.6}$$

10. THEOREM. *A group action of a finite group G on a set X induces a group homomorphism from G to the symmetric group S_X by $g \to \bar{g}$, where $\bar{g}$ is called the* permutation representation of G on X *and is denoted by the mapping $x \to g.x$.*

It's now possible to introduce two important lemmas that are fundamental for understanding Pólya's enumeration; while they're named after William Burnside, they're not due to Burnside himself but probably to Frobenius (1887).

2. LEMMA (BURNSIDE I). *The number of the G-orbits on the multiplicative finite group G acting on a finite set X is given by the average number of fixed points:*

$$|G\backslash\backslash X| = \frac{1}{|G|}\sum_{g\in G}|X_g| \tag{3.7}$$

where $X_g = \{x \in X,\ gx = x\}$ is the set of all fixed points of $g \in G$.

This lemma has an important generalization. In order to introduce it, it's possible to define a *weight function* $w : X \to R$ as a constant function for each G-orbit:

$$w(g.x) = w(x)\ \ \forall g \in G, \forall x \in X \tag{3.8}$$

where R is a commutative ring such that $\mathbb{Q}$ is a subring of R and G is a finite group acting on the finite set X. The weight of an orbit is equal to the weight of any of its element $w(G(x)) = w(x)$.

3. LEMMA (BARNSIDE II). *The sum of the weights of G-orbits is given by the average number of weighted fixed points:*

$$\sum_{u\in G\backslash\backslash X} w(u) = \frac{1}{|G|}\sum_{g\in G}\sum_{x\in X_g} w(x). \tag{3.9}$$

3.2.2 Pólya's theorem

This section will introduce one of the milestone results in the field of configuration-counting problems. The results are mainly given to Fripertinger ((Jedrzejewski 2006)).

In the whole section R is a commutative ring such that $\mathbb{Q}$ is a subring of it, X, Y are two finite sets and G is a finite group acting on X.

The set of functions $Y^X : X \to Y$ is called *set of configurations* from X; the group action of G on X induces an action of G on the set Y^X by $G \times Y^X \to Y^X$,

$$(g, f) \mapsto f \circ \bar{g}^{-1} \tag{3.10}$$

where $\bar{g}$ is a permutation representation of g acting on X and the weight function by $\omega : Y^X \to R$ on Y^X, induced by the weight function $h : Y \to R$ and defined as:

$$\omega(f) = \prod_{x \in X} h(f(x)). \tag{3.11}$$

The function ω is always constant on the G-orbits on Y^X; then for each $g \in G$:

$$\omega(gf) = \prod_{x \in X} h(f(\bar{g}^{-1}x)) = \prod_{x \in X} h(f(x)) = \omega(f). \tag{3.12}$$

It's called *cycle index* of an action G on X the polynomial $P_{(G,X)}$ of $\mathbb{Q}[t_1, \ldots, t_{|X|}]$ defined by:

$$P_{(G,X)}(t_1, \ldots, t_{|X|}) = \frac{1}{G} \sum_{g \in G} \prod_{k=1}^{|X|} t_k^{jk(\bar{g})} \tag{3.13}$$

where $jk(\bar{g})$ is the number of cycles of length k of the permutation $\bar{g}$ in its decomposition as a product of independent cycles.

11. THEOREM (PÓLYA'S ENUMERATION). *The sum of weights of G-orbits on Y^X is given by*

$$\sum_{u \in G \backslash\backslash Y^X} w(u) = \frac{1}{|G|} \sum_{g \in G} \prod_{k=1}^{|X|} \left(\sum_{y \in Y} h(y)^k \right) \tag{3.14}$$

where $jk(\bar{g})$ is as before.

The cycle index of the symmetric group S_n of the set $X = \{1, 2, \ldots, n\}$ of n elements is:

$$P_{(S_n,X)} = \sum_j \prod_k \frac{1}{jk!} \left(\frac{t_k}{k} \right)^{jk} \tag{3.15}$$

where the sum is verifying $\sum_{k=1}^{n} k_{jk} = n$ and is taken over all $j = (j_1, j_2, \ldots, j_n)$.

3.3 Classification of tempered chords

All the theory discussed above and expecially Pólya's theorem can be fruithfully used to enumerate chords in tempered-tuning systems made on n distinct notes, as described in paragraph 1.1.3. The results of this section are due to George Halsey and Edwin Hewitt and to Harald Fripertinger ((Jedrzejewski 2006)).

In order to numerically represents notes, it's important to define some concepts. It's called *pitch class (pc)* an integer number in $\mathbb{Z}_{12} = \{0, 1, \ldots, 11\}$ representing a note in a tempered-tuning system made of 12 steps, with the following association: $C = 0, C\sharp = 1, \ldots, B = 11$. Enharmonic equivalents (such as $F\sharp$ and $G\flat$) are numbered with the same integer. A set of pitch classes of cardinality k is called k-chord.

To enumerate k-chords, it's important to define the following:

- former set X is now identified with $\mathbb{Z}_{12}$;
- former group G is now one of the following[2]: cyclic $\mathcal{C}_n$, dihedral $\mathcal{D}_n$, or affine $\mathcal{A}_n$;
- a pitch class set corresponds to a characteristic function that maps the pitches in the set to 1 and other pitches to 0;
- $F(\mathbb{Z}_{12}) = Y^X$ is the sets of all pcsets;
- G acts of $F(\mathbb{Z}_{12})$ inducing an equivalence relation on sets made of k notes (k-chords);
- each k-chord is a subset of the Cartesian product X^k.

The definition of pcset given above can be generalized for any temperament made of n pitch classes. If G is a group acting on $F(\mathbb{Z}_{12})$, the pcset classes relatively to G are defined by the quotient set $F(\mathbb{Z}_{12}/G)$. If the acting group is the dihedral $\mathcal{D}_n$, the pcset classes $F(\mathbb{Z}_{12}/\mathcal{D}_n)$ are called *d-classes* ; if the acting group is instead the cyclic $\mathcal{C}_n$ then che pcset classes $F(\mathbb{Z}_{12}/\mathcal{C}_n)$ are called *c-classes* or *musical assemblies* .

[2]The case for the symmetric group $\mathcal{S}_n$ will not be discussed in this context.

Operations and group equivalence

Using some operations, it's possible to create equivalences under specific groups. It's called *transposition* the mapping T_a defined as:

$$T_a : \mathbb{Z}_n \to \mathbb{Z}_n \;\; T_a(x) = x + a \mod n \tag{3.16}$$

while it's called *inversion* the mapping I defined as:

$$I : \mathbb{Z}_n \to \mathbb{Z}_n \;\; I(x) = -x \mod n. \tag{3.17}$$

Finally, it's called *affine transformation* $M_{a,b}$ the following mapping:

$$M_{a,b} : \mathbb{Z}_n \to \mathbb{Z}_n \;\; M_{a,b}(x) = ax + b \mod n. \tag{3.18}$$

If two pcsets are reducible to each other by transposition they are equivalent under the cyclic group $\mathcal{C}_n$; if the are reducible to the same form by transposition or by inversion followed by transposition they are equivalent under the dihedral group $\mathcal{D}_n$. Finally, if the two pcsets are reducible to the same form by affine transformation the are equivalent under the affine group $\mathcal{A}_n$.

Computing cycle indexes

The number of k-chord classes is the number of G-orbits which is the coefficient of z_k in the cycle index for variables $t_k = 1 + z^k$.

The cycle index of the cyclic group $\mathcal{C}_n$ is given by the polynomial

$$P_{(\mathcal{C}_n,\mathbb{Z}_n)}(t_1, \ldots, t_n) = \frac{1}{n} \sum_{d|n} \varphi\left(\frac{n}{d}\right) t_{n/d}^d \tag{3.19}$$

where φ is the Euler's totient function [3].

The cycle index of the dihedral group $\mathcal{D}_n$ depends on the oddity of n and is given by:

[3] Briefly, the Euler's totient function φ for integer m is the number of positive integers not greater than and coprime to m.

$$P_{(\mathcal{D}_n,\mathbb{Z}_n)} = \begin{cases} \frac{1}{2}P_{(\mathcal{C}_n,\mathbb{Z}_n)} + \frac{1}{2}t_1 t_2^{(n-1)/2} & \text{if } n \text{ is odd} \\ \frac{1}{2}P_{(\mathcal{C}_n,\mathbb{Z}_n)} + \frac{1}{4}t_1^2 t_2^{(n-2)/2} + t_2^{n/2} & \text{if } n \text{ is even.} \end{cases}$$

The case of the affine group is more complicated, since it splits into two distinct cases:

1. $n = 2^a$: the cycle index of the affine group $\mathcal{A}_n$ is given by

$$\begin{aligned} P_{(\mathcal{A}_{2^a},\mathbb{Z}_{2^a})} &= \frac{1}{2^{(2a-1)}}\left(2^{2(a-1)-1}t_{2^a} + \sum_{i=1}^{a-1}\left(2^{2(i-1)} + \varphi(2^{i-1})2^{a-1}\right)t_{2^i}^{2^{a-i}} + \right. \\ &\left. \sum_{i=0}^{a-2}\varphi(2^i)\left(2^i t_1^{2^{a-i}} + 2^{a-1}t_1^2 t_2^{2^{a-i-1}-1}\right)\left(\prod_{k=1}^{i} t_{2^k}\right)^{2^{a-i-1}}\right) \end{aligned}$$

   for $a \geq 3$. For $a = 2$ and $a = 4$, however, there are smaller solutions:

$$\begin{aligned} P_{(\mathcal{A}_2,\mathbb{Z}_2)}(t_1,t_2) &= \frac{1}{2}(t_1^2 + t_2) \\ P_{(\mathcal{A}_4,\mathbb{Z}_4)}(t_1,t_2,t_3,t_4) &= \frac{1}{8}(t_1^4 + 2t_1^2 t_2 + 3t_2^2 + 2t_4). \end{aligned}$$

2. $n = p^a$, with p prime and $a \geq 1$: the cycle index of the affine group $\mathcal{A}_n$ is given by

$$\begin{aligned} P_{(\mathcal{A}_{p^a},\mathbb{Z}_{p^a})} &= \frac{1}{p^{2a-1}(p-1)} + \left(\sum_{i=1}^{a} p^{2(i-1)}(p-1)t_{p^i}^{p^{a-i}} + \right. \\ &\left. \sum_{i=0}^{a-1}\sum_{d|p-1} p^{i+\delta(d)(a-i)}\varphi(p^i d)t_1 t_d^{(p^{a-i-1}-1)/d}\left(\prod_{k=1}^{i} t_{p^k d}\right)^{p^{a-i-1}(p-1)/d}\right) \end{aligned}$$

   where $\delta(x) = 1$ if $x > 1$ and $\delta(1) = 0$.

Using the cycle indexes introduced above, it's finally possible to compute the number of assemblies from each class of k-chords in $\mathbb{Z}_{12}$; table 3.1 summarizes the obtained results and indicates the common name for each k-chord. Note that the problem is always symmetric with the center of ¿ in class 6: each chord of k notes has a complementary chord of $(12 - k)$ notes; the 12-chord is unique. For more details see (Jedrzejewski 2006).

Other typologies of classification are also possible; section 3.4.5 will show a classification of hexachords using combinatoriality.

Table 3.1: The total number of assemblies for any k in $\mathbb{Z}_{12}$.

	k-**chords**	**Group** $\mathcal{A}_n$	**Group** $\mathcal{C}_n$	**Group** $\mathcal{D}_n$
1	Unison	1	1	1
2	Intervals	5	6	6
3	Trichords	9	19	12
4	Tetrachords	21	43	29
5	Pentachords	25	66	38
6	Hexachords	34	80	50
7	Eptachords	25	66	38
8	Octochords	21	43	29
9	Ennachords	9	19	12
10	Decachords	5	6	6
11	Endecachords	1	1	1
12	Dodecachords	1	1	1
	Total	157	351	223

3.4 The pitch class set theory

This section will review the basic results of the pitch class set theory, following the approach presented by Allen Forte (Forte 1977).

3.4.1 Ordering

The pitch class sets defined in section 3.3 were unordered; if a set is also ordered it's called *tone row* (or *k-row* if its cardinality is k). A pitch class set is in *normal order* when, arranged in ascending order, it's also put in the more compact form by a cyclic permutation. Formally: let $A = \{A_0, A_1, \ldots, A_{k-1}\}$ be a pcset. For each cyclic permutation φ, the index vector from the permuted set $\varphi(A) = \{A_{\varphi(0)}, A_{\varphi(1)}, \ldots, A_{\varphi(k-1)}\}$ is defined by the vector $(u_1, \ldots, u_{k-1})$ with

$$u_{k-j-1} = A_{\varphi(j)} - A_{\varphi(0)} \bmod 12. \tag{3.20}$$

A given permutation φ_0 produces the normal order if the *compact number* $N(\varphi_0)$, or the number built on the coordinates of the index vector, is as small as possible:

$$N(\varphi 0) = u_1 \cdot 10^{k-2} + u_2 \cdot 10^{k-3} + \cdots + u_{k-2} \cdot 10 + u_{k-1}. \tag{3.21}$$

The number of transposition semitons of the normal order from the reference pcset is given by the index number $A_{\varphi_0(0)-A_0}$.

A pcset is in *prime form* (dihedral or Forte's prime forme) if its first integer is 0 and it's the most compact form among its inversion. Two pcsets are *equivalent* (or D_{12}-equivalent, where D_{12} is the dihedral group that acts on the set of all pcsets) if they are reducible to the same prime form by means of transposition or inversion followed by transposition.

3.4.2 Operations on pitch class sets

The *set matrix* of the prime form $A = \{A_0, A_1, \ldots, A_{k-1}\}$is a matrix U of size $k \times k$ as defined below:

$$U_{ij} = A_{i-1} + A_{j-1} \bmod 12. \tag{3.22}$$

1. PROPOSITION. *Given the pcset $A = \{A_0, A_1, \ldots, A_{k-1}\}$ of length k and the counting function 1_X (where $1_X = 1$ if X is true, $1_X = 0$ otherwise) if there is a number m such that*

$$k = \sum_{i,j} 1_{(U_{ij}=m)} \tag{3.23}$$

then $I_m(A) = T_m I(A) = A$; in other words, the pcset has the same pitch classes of its m-transposed inversion.

The *comparison matrix* of the pcset A given above is the matrix C of size $k \times k$ defined as:

$$C_{ij} = \text{sign}(A_{j-1} - A_{i-1}). \tag{3.24}$$

2. PROPOSITION. *Assuming that each comparision matrix is skew-simmetric ($C^T = -C$), the following cases hold:*

- inversion *: the comparison matrix of the inverse set $I(A)$ is the transposition matrix of C:*

$$C(I(A)) = C^T; \tag{3.25}$$

- retrograde *: the comparison matrix of the retrogade set $R(A)$ is the π-rotation of the elements of the matrix around its center C^R:*

$$C(R(A)) = C^R \tag{3.26}$$

where $C^R_{ij} = C_{k+1-i,k+1-j}$;

- retrograde-inversion *: the comparison matrix of the retrogade-inversion set $RI(A)$ is the codiagonal transposition of the matrix:*

$$C(RI(A)) = C^{RT} = C^{TR} \tag{3.27}$$

where $C^{RT}_{ij} = C_{k+1-j,k+1-i}$.

A pcset A^c is called *complement* of the pcset A if it's made of the elements of $\mathbb{Z}_{12}$ not contained in A; it's sometimes indicated by $A^c = \mathbb{Z}_{12}/A$. A set is said to be *self-complement* if both A and A^c are reducible to the same prime form.

3.4.3 Intervallic content

It's called *interval class* (ic) of two pcsets the function d that maps $\mathbb{Z}_{12} \times \mathbb{Z}_{12} \to \{0, 1, 2, 3, 4, 5, 6\}$ as follows:

$$d(x, y) = \begin{cases} |x - y| \bmod 12 & \text{if } |x - y| < 6 \\ -|x - y| \bmod 12 & \text{if } |x - y| \geq 6. \end{cases}$$

Given the $(k + 1)$-row $S = [S_0, \ldots, S_k]$ with $S_1 < \cdots < S_k$, it's called *derivation* of S the pcset D defined by:

$$D_j = d(S_j, S_{j+1}) = S_{j+1} - S_j \bmod 12 \tag{3.28}$$

for all $j = 0, \ldots, k-1$ and $D_k = S_0, \ldots, S_k$. It's also defined *iterated derivation* D^m the derivation applied recursively m times on S.

The *interval vector* (iv) of a pcset A is a 6-tuple representing all individual interval classes present in A. The first entry of the vector counts the number of the smallest interval (semitone), the second entry counts the second smallest and so on.

With *interval class content vector* (ivc) of two pcsets A, B is defined the 7-tuple $ivc(A, B) = [v_0, \ldots, v_6]$ where $v_0 = |A \cap B|$ and for $i = 1, \ldots, 6, v_i = |A \cap_k B|/2$. $A \cap_k B$ is the set of all pairs of intervals class k:

$$A \cap_k B = \{(x, y) \in A \times B, d(x, y) = k\}. \tag{3.29}$$

12. THEOREM (Z-RELATION). *Two pcsets A, B are said to be Z-related if they have the same interval vector ($iv(A) = iv(B)$) but they are not reducible to the same prime form.*

3.4.4 Similarity

Two pcsets A and B of the same cardinality m are *p-similar* ($\sim_p$) if there exists at least a common subset of cardinality $m - 1$ in the union of two representatives $\bar{A}$ and $\bar{B}$:

$$A \sim_p B \iff \exists C \;\; |C| = m - 1, C \subset \bar{A} \cup \bar{B}. \tag{3.30}$$

It's also useful to define the the *degrees* of similarity $\sim_0, \sim_1$ and $\sim_2$. Two pcsets of the same cardinality will be 0-similar if they have no equal values in the corresponding entries of the interval vector:

$$A \sim_0 B \iff \forall i, \; iv(A)_i \neq iv(B)_i \tag{3.31}$$

Two pcsets of the same cardinality will be 1-similar $A \sim_1 B$ if they have four equal corresponding entries in the inverval vector out of six and two inverted entries; in a similar way, they will be 2-similar if the have two equal

corresponding entries in the inverval vector out of six but two different entries.

Two sets classes A and B are said to be in *subset relation* ($\supset\subset$) if they have different cardinalities and one is included into the other:

$$A \supset\subset B \iff A \subset B \vee B \subset A. \tag{3.32}$$

The *set complex* $K(A, A^c)$ of the set class A is the set of all set classes B in subset relation with A or with its complement A^c:

$$B \in K(A, A^c) \iff B \supset\subset A \vee B \supset\subset A^c. \tag{3.33}$$

Finally, the *set subcomplex* $Kh(A, A^c)$ of a set class A is the set of all set classes B in subset relation with A and with its complement A^c:

$$B \in Kh(A, A^c) \iff B \supset\subset A \wedge B \supset\subset A^c. \tag{3.34}$$

3.4.5 Hexachordal combinatoriality and classification

Hexachords are special structures because of their intrinsic property of using half of the possible notes in $\mathbb{Z}_{12}$. For this reason they have been investigated a lot in term of *combinatoriality*. This paragraph will provide an overview of the basic results and will give different classifications of hexachords.

Combinatoriality is defined through the basic transformations on k-rows defined in section 3.3 and in paragraph 3.4.2: P (original row), I (inversion), R (retrograde), RI (retrograde of the inversion), T (transposition) and affine transformations $M5/7$ [4]. The transformations P, I, R and RI constitute a Klein group, whose multiplicative table is showed in table 3.2.

It's called *all-combinatorial* a k-row whose first hexachord forms a twelve-tone row with any of its basic transformations transposed. It's known that the second hexachord of all-combinatorial rows is always a transposition T_i with $i \neq 0$ of the first hexachord. The following theorem will define hexachordal all-combinatoriality τ; for more information see (Forte 1977).

[4] This transformations operate a multiplication of each element by 5 and 7 respectively; these are the only multiplicative transformations that preserve the cardiality of the original set in $\mathbb{Z}_{12}$.

Table 3.2: The multiplicative table for the Klein group made by P, I, R and RI.

P	P	I	R	RI
I	I	P	RI	R
R	R	RI	P	I
RI	RI	R	I	P

13. THEOREM (HEXACHORDAL τ-COMBINATORIALITY). *There are only six types of all-combinatorial row and they can be organized into four categories. If A if the first hexachord of the twelve-tone row S then:*

1. *τ_1-combinatoriality: A combines only with its transposition T_6: $S = A \cup T_6(A)$;*
2. *τ_2-combinatoriality: A combines with transpositions T_3 and T_9: $S = A \cup T_3(A) = A \cup T_9(A)$;*
3. *τ_3-combinatoriality: A combines with transpositions T_2, T_6 and T_{10}: $S = A \cup T_2(A) = A \cup T_6(A) = A \cup T_{10}(A)$;*
4. *τ_4-combinatoriality: A combines with all odd transpositions: $S = A \cup T_1(A) = A \cup T_3(A) = A \cup T_5(A) = A \cup T_7(A) = A \cup T_9(A) = A \cup T_{11}(A)$.*

4. LEMMA. *An hexachord A has the same interval vector of its complementary set A^c.*

A k-row is called *semi-combinatorial* if any of its transformations other than the retrograde can be transposed so that the first six notes are equivalent to the last six of the original set, even if in different order. The following theorem will define the categories of semi-combinatorial properties.

14. THEOREM (HEXACHORDAL SEMI-COMBINATORIALITY). *An hexachord A that is not all-combinatorial can have four categories of semi-combinatoriality:*

- *α-combinatoriality: if $A = A^c$ and $A \neq I(A)$;*

- *β-combinatoriality: if $A \neq A^c$ and $A = I(A)$;*
- *γ-combinatoriality: if $I(A) = A^c$ but they differ from A;*
- *ν-combinatoriality (*empty*): if $I(A) \neq A^c$ and they also differ from A.*

Orthogonality of hexachordal taxonomies

In order to know the total number of hexachords present in Z_{12} is also possible to apply the formula of *binomial coefficients*:

$$\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}. \tag{3.35}$$

With equation 3.35 is possible to compute how many different combinations of k elements can be created using n different elements. The total number of hexachords in Z_{12} is then given by:

$$\binom{12}{6} = \frac{12!}{6! \cdot (12-6)!} = \frac{479001600}{720 \cdot 720} = 924. \tag{3.36}$$

The classification in 924 classes is, however, not used; a more common taxonomy is found, among the others, in (Martino 1961) and in (Forte 1977) and is the one given in section 3.3 under the cyclic group $\mathcal{C}_n$; it reports 80 hexacords.

Looking carefully to the hexachords present in such a catalog it's possible to notice that some of them have the same interval vector; in other words, they are essentially *identical*.

It is then possible to apply another quotientation by means of the I transformation in order to create a smaller taxonomy in which all the interval vectors are different. After this operation, the number of hexachords shrinks to 35 classes E_I in which each class has a different interval vector; table 3.3 shows the new taxonomy.

It's really important to point out the following: combinatorial properties *propagates* from the 80-classes taxonomy to the 35-classes taxonmy. In other words, if an hexachord A belongs to a class of the original catalog with a given combinatorial property, that hexachord will belong to a class $e \in E_i$ that has the same combinatorial property.

35-class	**80-hexachords**	**Interval vector**	**Combinatoriality**
E_1	1	(5,4,3,2,1,0)	τ
E_2	2, 6	(4,4,3,2,1,1)	γ
E_3	3, 5, 7, 21	(4,3,3,2,2,1)	ν
E_4	4, 22	(4,3,2,3,2,1)	β
E_5	8, 20	(3,4,2,2,3,1)	γ
E_6	9, 18, 23, 54	(3,3,3,2,3,1)	ν
E_7	10, 15, 26, 36	(3,3,3,3,2,1)	ν
E_8	11	(3,4,3,2,3,0)	τ
E_9	12, 19	(4,2,2,2,3,2)	γ
E_{10}	13, 17, 27, 53	(3,3,2,2,3,2)	ν
E_{11}	14, 55	(3,2,4,2,2,2)	β
E_{12}	16, 37	(4,2,1,2,4,2)	β
E_{13}	24, 52	(3,2,3,4,2,1)	ν
E_{14}	25, 46	(3,2,3,4,3,0)	α
E_{15}	28, 51	(2,4,1,4,2,2)	γ
E_{16}	29, 45, 56, 63	(2,3,3,3,3,1)	ν
E_{17}	30, 35	(2,4,2,4,1,2)	γ
E_{18}	31, 38, 49, 50	(3,2,2,3,3,2)	ν
E_{19}	32, 43, 64, 66	(2,3,3,2,4,1)	ν
E_{20}	33, 58	(2,3,4,2,2,2)	β
E_{21}	34, 40	(3,2,2,4,3,1)	γ
E_{22}	39, 44, 59, 62	(3,1,3,4,3,1)	ν
E_{23}	41, 48	(3,2,2,2,4,2)	γ
E_{24}	42, 67	(2,3,2,3,4,1)	β
E_{25}	47	(4,2,0,2,4,3)	τ
E_{26}	57, 61	(2,2,5,2,2,2)	γ
E_{27}	60, 65	(2,2,4,3,2,2)	β
E_{28}	68, 79	(1,4,2,4,2,2)	γ
E_{29}	69, 75	(1,4,3,2,4,1)	γ
E_{30}	70, 77	(2,2,3,4,3,1)	γ
E_{31}	71	(1,4,3,2,5,0)	τ
E_{32}	72, 78	(2,2,4,2,2,3)	γ
E_{33}	73, 74	(2,2,4,2,3,2)	β
E_{34}	76	(3,0,3,6,3,0)	τ
E_{35}	80	(0,6,0,6,0,3)	τ

Table 3.3: Taxonomy of hexachords in 35 classes by means I.

While in the new taxonomy there aren't classes with the same interval vector it's still possibile to apply another quotientation by means of the affine transformations $M5/7$. This leads to a new classification made of 26 classes F_i; table 3.4 shows the new shrinked taxonomy.

Looking at the interval vectors of classes F_i it's possible to notice that after the application of $M5/7$ the entries number 1 and 5 are *exchanged*. For example, hexachord 69, whose interval vector is $(1, 4, 3, 2, 4, 1)$ geneteres (under $M5$) hexachord 2, whose interval vector is $(4, 4, 3, 2, 1, 1)$. This new taxonomy, then, assumes as equivalent all classes whose interval vector are equal except for the exchange of entries 1 and 5.

Amazingly enough, also in this case the combinatorial properties preserve. This is an indirect proof that quotientations by basic transformations are correct; all these hexachordal taxonomies are said to be *orthogonal*.

3.5 Transformational theory

The results of this section are mainly due to D. Lewin; he was the first to introduce the notion of *generalized interval system* (GIS) in (Lewin 2007).

3.5.1 Generalized interval system

Given a set X with finite elements and a multiplicative group G of intervals on X, it's possible to define a *generalized interval system* as a triple (X, G, int) where int is the function $X \times X \to G$ such as:

- $int(x, y) \circ int(y, z) = int(x, z) \;\; \forall x, y, z \in X$;
- $\forall x \in X, \forall g \in G$ there exists a single value $y \in X$ such that $int(x, y) = g$.

With 1_X is defined the *characteristic function* of the non-empty pcset X, such that:

$$1_X(u) = \begin{cases} 1 & \text{if } \; u \in X \\ 0 & \text{otherwise.} \end{cases}$$

The function $1_X^*(u) = 1_X(-u)$ is called the *adjoint* function of 1_X.

26-class	**35-class**	**80-hexachords**	**Interval vector**	**Comb.**
F_1	E_1, E_{31}	1, 71	(5,4,3,2,1,0)	τ
F_2	E_2, E_{29}	2, 6, 69, 75	(4,4,3,2,1,1)	γ
F_3	E_3, E_{19}	3, 5, 7, 21, 32, 43, 64, 66	(4,3,3,2,2,1)	ν
F_4	E_4, E_{24}	4, 22, 42, 67	(4,3,2,3,2,1)	β
F_5	E_5	8, 20	(3,4,2,2,3,1)	γ
F_6	E_6	9, 18, 23, 54	(3,3,3,2,3,1)	ν
F_7	E_7, E_{16}	10, 15, 26, 29, 36, 45, 56, 63	(3,3,3,3,2,1)	ν
F_8	E_8	11	(3,4,3,2,3,0)	τ
F_9	E_9, E_{23}	12, 19, 41, 48	(4,2,2,2,3,2)	γ
F_{10}	E_{10}	13, 17, 27, 53	(3,3,2,2,3,2)	ν
F_{11}	E_{11}, E_{33}	14, 55, 73, 74	(3,2,4,2,2,2)	β
F_{12}	E_{12}	16, 37	(4,2,1,2,4,2)	β
F_{13}	E_{13}, E_{30}	24, 52, 70, 77	(3,2,3,4,2,1)	ν
F_{14}	E_{14}	25, 46	(3,2,3,4,3,0)	α
F_{15}	E_{15}	28, 51	(2,4,1,4,2,2)	γ
F_{16}	E_{17}, E_{28}	30, 35, 68, 79	(2,4,2,4,1,2)	γ
F_{17}	E_{18}	31, 38, 49, 50	(3,2,2,3,3,2)	ν
F_{18}	E_{20}	33, 58	(2,3,4,2,2,2)	β
F_{19}	E_{21}	34, 40	(3,2,2,4,3,1)	γ
F_{20}	E_{22}	39, 44, 59, 62	(3,1,3,4,3,1)	ν
F_{21}	E_{25}	47	(4,2,0,2,4,3)	τ
F_{22}	E_{26}	57, 61	(2,2,5,2,2,2)	γ
F_{23}	E_{27}	60, 65	(2,2,4,3,2,2)	β
F_{24}	E_{32}	72, 78	(2,2,4,2,2,3)	γ
F_{25}	E_{34}	76	(3,0,3,6,3,0)	τ
F_{26}	E_{35}	80	(0,6,0,6,0,3)	τ

Table 3.4: Taxonomy of hexachords in 26 classes by means $M5$.

The *interval function* $ifunc_{(X,Y)}$ for two non-empty pcsets X and Y is defined as the convolution of the characteristic functions $1^*_X \star 1_X$:

$$ifunc_{(X,Y)}(i) = \sum_j 1^*_X(j) \cdot 1_Y(i-j) = \sum_k 1_X(k) \cdot 1_Y(i+k). \tag{3.37}$$

Using the interval function is possible to find how many times the note k in the pcset X has its i-transpose in Y. In this context, the *interval vector* is the 12-tuple whose entries are represented by the functions $ifunc_{(X,Y}(i)$ for $i = 0, \ldots, 11$. Finally, with *injection function* $inj_{(X,Y)}(f)$ it's defined the number of elements $x \in X$ relative to the transformation $f(x) \in Y$ such that:

$$inj_{(X,Y)}(f) = \sum_{x \in X} 1_{(f(x) \in Y)}. \tag{3.38}$$

The following theorems represent important milestones in the trasformational theory. They will be presented without proof; for more information see (Jedrzejewski 2006) and (Lewin 2007).

15. THEOREM (TRANSLATION PROPERTY). *If T_i is the translation if i and X and Y are two non-empty pcsets then*

$$inj_{(X,Y)}(f) = ifunc_{(X,Y)}(i). \tag{3.39}$$

16. THEOREM. *If A is a pcsets whose cardinality is 6 (hexachord) and $|A| = |A^c|$, then for all bijections f:*

$$inj_{(A,A)}(f) = inj_{(A^c,A^c)}(f) \tag{3.40}$$

and in particular

$$ifunc_{(A,A)} = ifunc_{(A^c,A^c)}. \tag{3.41}$$

17. THEOREM. *If A is a pcsets whose cardinality is 6 (hexachord) and $|A| = |A^c| = 6$, then for all bijections f:*

$$inj_{(A,A)}(f) + inj_{(A^c,A^c)}(f) = 6 \tag{3.42}$$

and in particular

$$ifunc_{(A,A_c)} = 6 - ifunc_{(A,A)}. \tag{3.43}$$

3.5.2 Relations with the discrete Fourier transform

There are some important relations between the characteristic function and the discrete Fourier transfom (DFT) [5].

It's indeed possible to define the Fourier transform of the characteristic function such as:

$$\mathcal{F}(1_X)(v) = \frac{1}{12}\sum_u 1_X(u)\cdot e^{\frac{-2\pi uv}{12}}. \tag{3.44}$$

For the theorem of convolution (see (Oppenheim and Schafer 2009)), it's known that a convolution in time domain corresponds to a pointwise multiplication in frequency domain. For this reason, the DFT of the interval function is the pointwise multiplication of the complex conjugates of $\mathcal{F}(1_X)$ with $\mathcal{F}(1_Y)$:

$$\mathcal{F}(X,Y) = \mathcal{F}(\bar{1}_X)\cdot\mathcal{F}(1_Y). \tag{3.45}$$

The following important results have been proved by D. Lewin in (Lewin 2007).

18. THEOREM. *If the Fourier transform $\mathcal{F}(X,Y)(i)$ of the interval function of two non-empty pcsets X and Y is equal to zero for $i = 1,\ldots,11$ then the interval function is constant.*

19. THEOREM. *If the Fourier transform $\mathcal{F}(X,Y)(i)$ of the interval function of two non-empty pcsets X and Y is equal to zero for all i excepts than 0 and 6, then the interval function is made of alternate entries:*

$$ifunc_{(X,Y)} = (p,q,p,q,p,q,p,q,p,q,p,q) \tag{3.46}$$

where p and q are integers.

20. THEOREM. *If X, Y, Z are non-empty pcsets, then $ifunc_{(X,Y)} = ifunc_{(X,Z)}$ if and only if the Fourier transforms $\mathcal{F}(1_X)$ or $\mathcal{F}(1_Y - 1_Z)$ are always equal to zero for any i.*

[5] More information about the DFT can be found in paragraph 4.3.

3.5.3 Riemannain transformations and K-nets

Riemann's theory has been introduced in paragraph 1.1.2: in his work he developed a new analytical approach to musical theory defining three transformations on his table of tonal relations. These transformations are *Parallel, Relative* and *Leittonwechsel* (P, R, L) ; they act on triads (pcsets of cardinality 3) and maintain two fixed points. A short review of them will be given below:

- *Parallel*: exchanges major and minor triads and it's defined as $[a, a + 4, b] \longleftrightarrow_P [a, a + 3, b]$;
- *Relative*: exchanges a major triad with its relative minor triad and it's defined as $[a, b, b + 3] \longleftrightarrow_R [a - 3, a, b]$;
- *Leittonwechsel*: exchanges a major triad with the minor triad located a major third up and it's defined as $[a, b, b + 3] \longleftrightarrow_R [a - 3, a, b]$.

It's possible to define new transformations as composition of the three discussed above:

- *subdominant* (S): transforms a major triad to a major triad transposed a fifth up and a minor triad to a minor triad transposed a fifth down; it's defined as $S = R \circ L$;
- *dominant* (D): transforms a minor triad to a minor triad transposed a fifth up and a major triad to a major triad transposed a fifth down; it's defined as $D = L \circ R$.

The three fundamental transformations P, L and R generate a group isomorphic to the dihedral group of order 24 created by the translations $T_n : x \to x + n \mod 12$ and the inversions $I_n : x \to -x + n \mod 12$.

H. Klumphenhouwer showed in (Jedrzejewski 2006) that any set of pitch classes can be represented as a network (called *Klumphenhouwer network* or K-net) of T and I transformations. This new perspective focuses on the structural morphology and on the logical progressions of chords interpreting them as graphs in wich pitch classes are linked by the T_n and I_n transformations. Figure 3.2 depicts the K-net for the three pcsets $\{0, 1, 4, 6, 7\}$, $\{0, 4, 6, 7, 10\}$ and $\{0, 1, 6, 7, 10\}$ showing how they actually can be derived using the T_3 transformation.

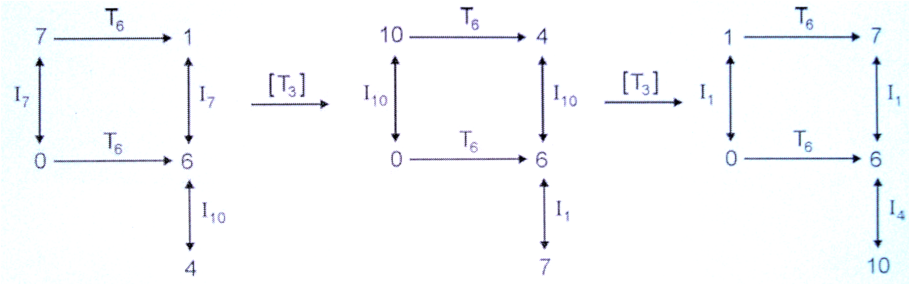


Figure 3.2: A k-net showing the transformations of the pcsets $\{0, 1, 4, 6, 7\}$, $\{0, 4, 6, 7, 10\}$ and $\{0, 1, 6, 7, 10\}$.

3.6 Selected topics

3.6.1 Hexachords and trichordal generators

Regular subdivisions of twelve-tone rows have been studied by different theoreticians such as, among the others, Donald Martino in his *The source set and its aggreate formations* (1961) (Martino 1961). The problem falls into the more general category of *musical mosaicing*, that will not be discussed in this context; for more information see (Jedrzejewski 2006).

The specific case of subdividing the twelve-tone row into four chords of three notes (*trichords*) had many connections with compositional designs thanks to composers such as Francesco Valdambrini , Peter Schat (Schat 1993) and Steve Rouse (Rouse 1984). The following results are due to Rouse; his point of view is particularly interesting because he studied the relations between twelve-tone rows (also called *aggreates*), hexachords and trichords understanding their combinatorial properties.

In the following discussion, the sets are numbered using Forte's catalog in which there are 12 trichords and 50 hexachords (see (Forte 1977)).

Each hexachord can be divided into 10 pairs of trichords; figure 3.3 shows the composition of hexachord $H = [0, 1, 2, 3, 4, 5]$ into the 10 trichordal couples.

If an hexachord is made of two trichords belonging to the same class (ie. with the same prime form) then it has a *single* generator , otherwise it has a

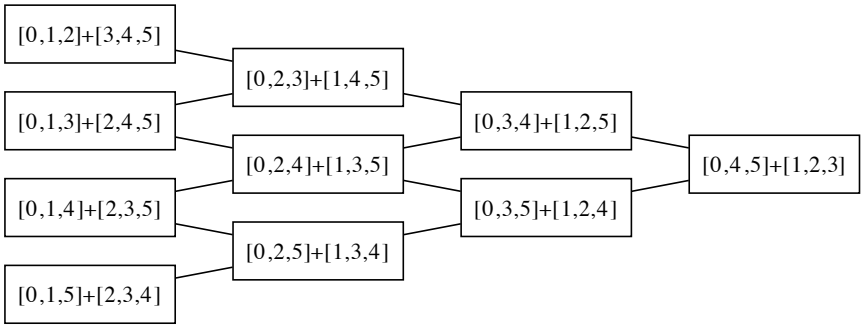


Figure 3.3: The decomposition of hexachord $H = [0, 1, 2, 3, 4, 5]$ into the generating couples of trichords.

dual generator . Rouse presents three important tables showing the following information:

- given an hexachord H, all the trichordal generators ;
- given two trichords T_1 and T_2, all the hexachords that they generate;
- a mixed table with the hexachords in the horizontal axis and the trichords in the vertical axis.

The possible trichordal generators are 78: the 12 existing trichords are coupled such as $12 + 11 + 10 + \cdots + 1 = 78$. It's possible, however, that some generators mix up in different ways (ie. using different transformations): in that case Rouse uses the power symbol to indicate the number of occurences (for example $12 - 25^2$ is used to indicate the trichords 12 and 15 combine in two different modes[6]).

Ttable 3.5 lists the beginnig of the hexachordal point of view (corresponding to Rouse's first table); table 3.6, on the other hand, shows the trichordal

[6]To improve readibility, the trichord $[0, x, y]$ will be indicated simply as xy; for example $[0, 1, 2]$ becomes 12.

Table 3.5: Hexachords and their trichordal generators.

Hex.	**Prime form**	**Int. vector**	**Single generators**	**Dual generators**
1	012345	543210	12, 13, 14, 24	12-15^2, 13-14^2, 13-25^2
2	012346	443211	14	12-13, 12-16, 12-26, 13-24, 13-25, 13-26, 13-36, 14-15, 24-25
*Z*3	012356	433221	25	12-13, 12-16, 13-14, 13-15, 13-16, 14-15, 14-36, 15-25, 24-26
*Z*36	012347	433221		12-13, 12-16, 13-37, 13-25, 13-27, 13-36, 13-37, 14-15, 14-16, 24-26
...				

point of view (corresponding to Rouse's second table).

The information provided by Rouse's tables is really interesting. Nonetheless, the tables only list *abstract* forms of trichords and hexachords without giving any information about the real transformations applied to the sets. For example, hexachord 1 is made by generator 12-15 in two ways:

- $012ab3$, made by $12 + T_{10}(15)$;
- $01243b$, made by $12IT_4(15)$.

The *constructive* information is really important for actual use since its computation can be really complex. There are softwares that can compute actual forms of the generators[7]; table 3.8 shows some of the actual transfor-

[7] A software has been written by the author of this work; it's called *OpenMosaic* and can be found online.

mations for the generatators of hexachord $[0, 1, 2, 3, 4, 5]$.

Compositional use of hexachords

It's possible to create sequences of hexachord using the musical principle of *modulation* : an hexachord is followed by another one that shares common generators. This method guarantees continuity of the hexachordal level while changes the trichordal level and has been used from composers in order to create harmonic successions. Table 3.7 illustrates this process showing common generators; the same hexachords are also depicted in figure 3.4, in which the *circularity* of the process appears more clearly.

Table 3.6: Trichords and generated hexachords

Trichords	**Generated hexachords**
12-12	1, Z4, Z6, 7
12-13	2, Z3, Z11, Z12, Z43, 18
12-14	Z36, 5, 15, Z17, Z44
12-15	12, Z372, Z382, 14, 16
...	

It's also possible to create hierarchies of hexachords depending on the number of shared trichords. Figure 3.5 illustrates this process for trichords $[0, 1, 2]$ and $[0, 1, 5]$.

Table 3.7: A hierarchy of hexachords by means of shared generators.

Modulated hexachords	**Shared generators**
6-2 / 6-Z3	12-13, 12-16, 14-15
6-Z3 / 6-5	12-16, 13-15, 13-16
6-5 / 6-Z12	13-16, 15-37, 25-27
...	

Trichordal mosaics

The following theorems, also due to Rouse, will define the *trichordal mosaics* and will give some important properties.

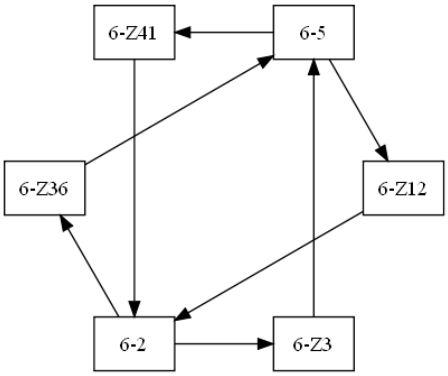


Figure 3.4: An example of modulation of hexachords.

21. THEOREM. *A* trichordal mosaic *is a set of four trichords that combine into an aggregate. Let A be an aggreate and X, Y, Z, W trichords; there are five different types of mosaics:*

- $A = X \cup X \cup X \cup X$*: the aggregate made by four identical trichords;*
- $A = X \cup X \cup X \cup Y$*: the aggregate made by three identical trichords and one different;*
- $A = X \cup X \cup Y \cup Y$*: the aggregate made by two couples of identical trichords;*
- $A = X \cup X \cup Y \cup Z$*: the aggregate made by three different thrichords, the first being used twice;*
- $A = X \cup Y \cup Z \cup W$*: the aggregate made by four different trichords.*

22. THEOREM (TRIANGULAR RELATION). *In any trichordal mosaic, the combination of vertical or diagonal pairs of trichords produces either the original hexachord or a pair of derived heaxhord. Derived hexachords will always be either a dual*

Table 3.8: Actual generators for hexachord [0, 1, 2, 3, 4, 5].

Forms of [0, 1, 2, 3, 4, 5]	**Actual generators**
012345	t0(12) t3(12)
0129ab	t0(12) t9(12)
012543	t0(12) it5(12)
012ba9	t0(12) itb(12)
012ab3	t0(12) ta(15)
01243b	t0(12) it4(15)
012b34	t0(12) tb(45)
0123ba	t0(12) it3(45)
013542	t0(13) it5(13)
013245	t0(13) t2(23)
...	

representation of a single, non Z-related hexachord, or a pair of Z-related hexachords because they are necessarily complementary.

There are 3081 ($1+2+3+\cdots+78 = 3081$) couples of trichordal generators, but only some of them can be used to generate aggregates. The combinatorial computation of these couples is complex and is possible by means of specialized algorithms.

Compositional use of mosaics

Musically speaking, one of the most interesting possibility of trichordal mosaics is creating *sequences* of mosaics that preserve the hexachordal level while changes the trichordal level. With this approach is possible to control the *color* of the harmony used, while changing the actual notes.

Figure 3.6 shows a sequence of trichordal mosaics; the hexachord 6-2 propagates in the horizontal level from aggregate to aggreate on the horizontal level, while vertical and diagonal hexachords always change. The sequence as a *generation period* that depends on the number of thricordal gen-

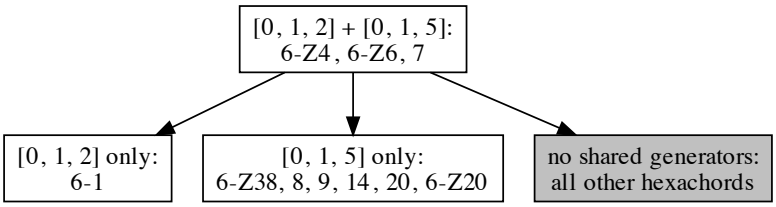


Figure 3.5: An example hierarchy of hexachords depending on shared trichords.

erators of the horizontal hexachord. It's possible, however, to concatenate several sequences changing the direction of the generation from horizontal to either horizontal or vertical.

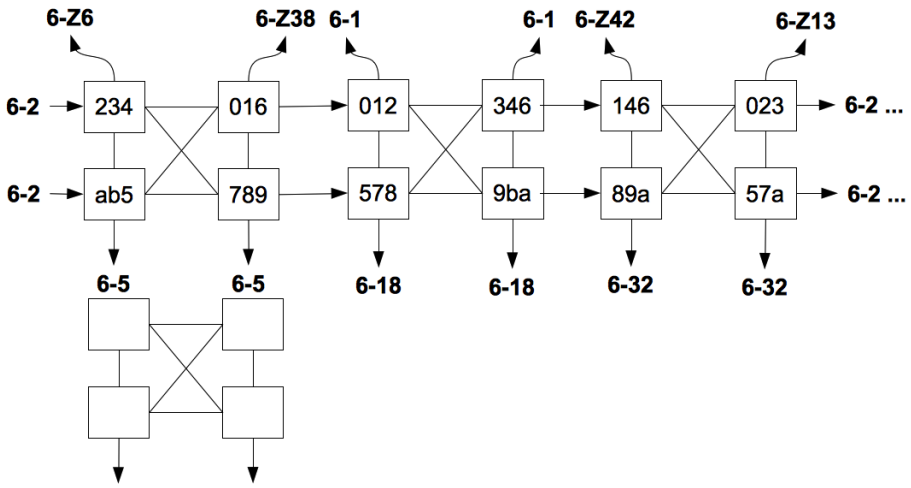


Figure 3.6: An example device for trichordal mosaics

3.7 Summary

In this chapter, the main results coming from the application of algebraic methods to music have been showed.

At the beginning of the chapter, a general review of the needed theoretical background has been given from the concept of *group action* to the Pólya's enumeration theory. These results have been used to classify and enumerate chords, a problem that received always a lot of attention from composers.

Then, pitch class set theory has been illustrated and the main theorems and results have been provided. The concepts of intervallic content and similarity have been discussed and a short review of combinatoriality of hexachords has also been given.

The transformational theory has been then presented, following David Lewin's approach. The concept of generalized interval system has been presented and its relations with the discrete Fourier transform have been showed. A short presentation of k-nets has also been given.

At the end of the chapter, the trichordal mosaics have been presented showing also some possible compositional applications.

Chapter 4

The theory of sound-types

The best way to predict the future is to invent the future.

Dennis Gabor

Abstract

Sound-types are a new method to represent and manipulate sounds in a quasi-symbolic way by means of low-level features and subsequent analysis stages. After the presentation of the basic ideas, a full analysis-synthesis framework and some applications will be shown.

4.1 The levels of representation

Symbolic representations described in chapters 2 and 3 have been extensively used in latest developments of musical theory. Algebraic approaches, in particular, entered the compositional process and changed the way musical creation is done.

As showed in chapter 1, however, all these representations have been applied to symbolically represent objects that are already symbolic. Musical notation, in fact, is configured to handle notes, chords, rests and other musical elements as symbols to be decoded. Nonetheless, music is not completely represented by the score: it exists in the final stage of *performance*[1]. Is it possible to apply symbolic representations on musical audio signals in the stage of performance? In other words, is it possibile to describe musical signals in a symbolic way?

[1]While music *can exist* at the statical level of the score, this chapter will only deal with *played* music, considering it as an acoustical phenomenon.

Music can be described in many ways: it can be viewed as a time-varying *signal* and can be described by expressing the evolution of its physical properties over time. Music can be also viewed as a *symbolic system* exploiting relationships between sonic-objects[2] and can be described by a formal language able to express these relationships over time[3].

Common approaches for music description generally take into account the different points of view by selecting a particular degree of abstraction in the domain of the representation: either they rely on the *signal level*, either on the *symbolic level* or on a fixed mixture of both[4]. The latter case is generally known as *mid-level* representation: this term is used in the computer audition community to indicate intermediate modelings of hearing usually based on perceptual criteria; see (Ellis and Rosenthal 1995).

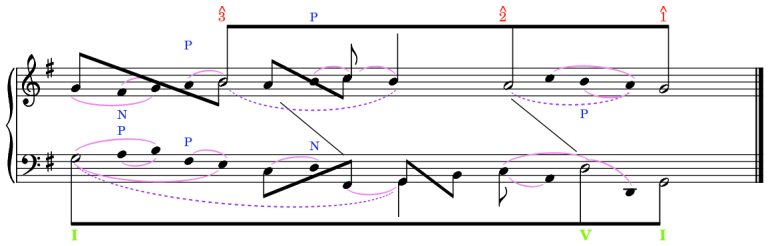


Figure 4.1: An example of symbolic-level representation; this represents a Schenkerian analysis in which symbols generically represent musical entities with repetitions and hyerarchies.

While signal-level representations are computationally efficient, invertible[5] and express some physical properties associated to the signal, they lack in abstraction and usually do not provide much of information about hierarchies, formal relationships between sonic-objects and so forth; they are

[2]With this expression, intuitively, it is referred any kind of event that appears in the musical flow; a precise definition of sonic-objects is exactly the scope of any representation.

[3]Understanding the formal properties of music can also involve perceptual aspects and memory, that will not be considered here.

[4]In (Vinet 2003) four different levels of representation are introduced, from specific to abstract: *physical representations*, *signal representations*, *symbolic representations*, *knowledge representations*; physical and knowledge represenations are beyond the scope of this research.

[5]*Invertibility* is the possibility to go back to the signal domain from the representation itself.

unable to manipulate concepts other than the *basis* of the analysis itself (such as sinusoids). A more detailed overview of signal-level representations will be given in section 4.3.

Symbolic-level representations can express complex relationships and hierarchies (structure analysis, repeated patterns, etc.) but are inefficient, non-invertible and are hardly related to the physical nature of sound. A typical example is depicted in figure 4.1: it represents a Schnkerian analysis where the symbols encode musical entities with repetitions and hierarchies. Symbolic-level representations are often based on *logical rules* that cannot always be verified by a computational model. Moreover, the underlying logic generally assumes statically predefined concepts that are *imposed* onto the signal. The form of representations described in chapters 2 and 3 are all examples of this level.

Mid-level representations try to address the issue related to the lack of generality by focusing on *relatively simple* concepts that are, however, more abstract than the bases of the analysis. These concepts are usually based on perceptual criteria related to the low-level hearing and are situated in between the constraints imposed on them by lower and higher levels. The power of this kind of representations stands in the fact that they are usually invertible and that the logical rules they involve are generally verifiable by some models related to perception. Mid-level representations, moreover, belong to a full *network* of representations spanning between the signal level and the symbolic level; it is always possible to create new types of mid-level representations to express desired concepts. Nevertheless, the concepts are *imposed* onto the signal from the representation itself in this case as well.

4.1.1 A new representation method

All the representation levels discussed so far have a common feature: a *fixed degree of abstraction*. In other words, they focus on a particular point of view and are not scalable: once a representation level has been selected it is not possible to go smoothly to another level.

Most of them, moreover, impose *their own* concepts onto the signal: each representation models the signal with its own concepts, even if they are completely irrelevant to *that* particular signal; figure 4.2 roughly depicts the de-

scribed ideas.

Figure 4.2: The levels of representation.

The main purpose of this chapter is to propose a connection between the signal and the symbolic-level by defining a new representation method based on specific signal processing techniques able to retrieve information from a signal and model that information statistically to find *salient* properties.

4.1.2 Properties for a representation

Defining a representation method for music and musical signals involves the establishment of essential properties that the representation must satisfy. Many years of research have been devoted to such a task in the field of *cognitive musicology*, a branch of cognitive sciences focused on the modeling of musical knowledge by means of computational methods (Laske et al. 1992); among the main researchers of the field there are Otto Laske, Mira Balaban, Bernard Bel, Francis Courtot, Fred Lerdahl and Ray Jackendoff.

The following is a partial list of important properties for musical representations arised from the examination of the literature in the field, organized by researcher:

- Francis Courtot in (Laske et al. 1992):
  - *subdivision of the objects in simple and complex*: music is necessarily built of elements with various degrees of complexity;
  - *horizontal and vertical associations between objects*: in music it is important to be able to express both time and hierarchy;

  - *heuristics to produce new objects*: as a generative language, music can create new objects;
  - *rules should be learned from the context*: not all the rules in music are defined from the beginning and some of the can arise during musical creation;

- Bernard Bel, in (Bel 1989):
  - *mapping between symbolic (prescriptive) and numeric (descriptive) representations*: this property is related to the possibility of describing music both as a score and as performance;
  - *terminal objects of the language have acoustical properties*: in a representaiton conceived for music it should be possibile to listen to the atoms of the language.

- Mira Balaban, in (Laske et al. 1992):
  - *musical operators are needed*: music performs transformation on its objects (such as transposition, time shifting, etc.);
  - *support for incomplete description*: many elements in music cannot be properly described.

The key-points of these properties are three. First, the representation should handle simple and complex objects. Second, the relations among the objects should be hierarchical and temporal. Third, atomic elements should have an *acoustical* counterpart.

Section 4.5 will present a new method for music representation called the *theory of sound-types*. The basic features of this representation have been defined keeping into consideration the properties given above. The core idea is to represent music using *common entities* that can be instantiated into specific elements. These common entities act as classes of equivalences for sounds and can therefore considered as *types*; the types are instantiated in time and relations and operations are defined over them.

The general theoretical background of the approach presented here takes its inspiration from simple type theory and from the *System F* by the french logician J.Y. Girard. Theory of sound-types, nontheless, rely on some signal

processing techniques and on some statistical techniques. The comprehension of all the background is needed in order to understand the theory. For this reason, following sections will introduce the basics of required fields.

Finally, a theoretical formulation of the proposed representation method will be given through the *sound-types transform* and a real implementation will be presented, showing how this method is a full framework for sound analysis and synthesis.

4.2 Inspiration: simple type theory

In order to avoid some set-theoretical paradoxes such as the one about classes of all classes that are not members of themselves, Bertrand Russell proposed in 1908 a logic now known as *ramified theory of types*; in the twenties, this theory has been simplified by the addition of the axiom of *reducibility* becoming the so-called *simple type theory*. In 1940, Alonzo Church (Church 1985) formulated simple type theory in terms of functions creating a special notation called λ-notation. Subsequently, this elegant formulation has been expanded in many ways becoming one of the most influential theories on modern computer science and on general theory of functions. There are many variants of simple type theory; the presentation given here is a version of Church's theory and is due to Farmer.

Simple type theory syntax is made of two principal objects: *types* and *expressions* . The former is a nonempty set of values and is used to build expressions, to validate them by value and to *restrict* the scope of variables. The latter, instead, denotes values including true and false values and behaves like terms and formulas in first-order logic.

A type of simple type theory is defined by the following formation rules:

T1. i is the type of individuals;

T2. $\star$ is the type of truth values;

T3. if α, β are types, then $\alpha \rightarrow \beta$ is the type of functions from elements of type α to elements of type β.

Rules T1 and T2 define he so-called *atomic* types while rule T3 defines *compound* types. The logical symbols of simple type theory are defined as

follows:

1. *function application*: @
2. *funcion abstraction*: λ
3. *equality*: $=$
4. *definite description*: ι
5. an infinite set of symbols called *variables*: ν

it is now possible to define a *language* of simple type theory as the ordered pair $L = \langle C, \phi \rangle$ where:

1. C is a set of symbols called *constants*;
2. $\nu \cup C = \oslash$ (the sets are disjoint);
3. $\phi : C \rightarrow \tau$ is a total function, where τ is a set of types of simple type theory.

In other words, a language is a set of symbols with types that have been *assigned*. It is now possible to define an expression of the language L with another set of formation rules:

E1. if α is a type and $x \in \nu$, then $x : \alpha$ is an expression of type α (*variable*);

E2. if $c \in C$, then c is an expression of type $\phi(c)$ (*constant*);

E3. if A is an expression of type α and F is an expression of type $\alpha \rightarrow \beta$, then $F@A$ is an expression of type β (*function application*);

E4. if $x \in \nu$, α is a type and B is an expression of type β then $\lambda x : \alpha.B$ is an expression of type $\alpha \rightarrow \beta$ (*function abstraction*);

E5. if E_1 and E_2 are expressions of type α, then $E_1 = E_2$ is an expression of type $\star$ (*equality*);

E6. if $x \in \nu$, α is a type and A is an expression of type $\star$, then $\iota x : \alpha.A$ is an expression of type α (*definite description*).

All expressions, here, are differentiated by type and not by *form*. For example, an *individual constant* of L is a constant $c \in C$ such that $\phi(c) = \iota$; a *formula* of L is an expression of type $\star$ while a *predicate* of L is an expression of type $\alpha \rightarrow \star$.

4.2.1 Models for simple type theory

For the languages of simple type theory, like for first-order languages, it is possible to define a semantics based on models. A *standard model* for a language $L = \langle C, \phi \rangle$ of simple type theory is an ordered triple $M = \langle D, E, I \rangle$ such that:

1. $D = \{D_\alpha : \alpha \in \tau\}$ is a set of nonempty domains;
2. $D_\star = \{true, false\}$;
3. for $\alpha, \beta \in \tau, D_{\alpha\to\beta}$ is the set of *all* functions from D_α to D_β;
4. $E = \{e_\alpha : \alpha \in \tau\}$ is a set of values such that $e_\alpha \in D_\alpha, \forall\alpha \in \tau$ (e_α is called the *canonical error* for α);
5. I maps each $c \in C$ to a member of $D_{\tau(c)}$.

Given a model $M = \langle D, E, I \rangle$ for a language of simple type theory, the *variable assignment* into M is a function ψ that maps each variable expression $x : \alpha$ to a member of D_α. Given a variable assignment ψ into M, an expression $x : \alpha$ and $d \in D_\alpha$, let $\psi(x : \alpha \to d)$ be a variable assignment $\psi\prime$ into M such that $\psi(x : \alpha) = d$ and $\psi\prime(v) = \psi(v), \forall v \neq x : \alpha$. The *valuation function*, then, is the binary function V^M that, for all variable assignments ψ and all expressions E of L, satisfies the following conditions:

1. if $E = x : \alpha$, then $V^M_\psi(E) = \psi(x : \alpha)$;
2. if $E = C$, then $V^M_\psi(E) = I(E)$;
3. if E is of the form $F@A$, then $V^M_\psi(E) = V^M_\psi(F)V^M_\psi(A)$;
4. if E is of the form $\lambda x : \alpha.B$ with B of type β, then $V^M_\psi(E)$ is the function $f : D_\alpha \to D_\beta$ such that $\forall d \in D_\alpha, f(d) = V^M_{\psi(x:\alpha\to d)}(B)$;
5. if E is of the form $E_1 = E_2$ and $V^M_\psi(E_1) = V^M_\psi(E_2)$, then $V^M_\psi(E) = true$; otherwise $V^M_\psi(E) = false$;
6. if E is of the form $Ix : \alpha.A$ with A of type α and there is a unique $d \in D_\alpha$ such that $V^M_{\psi(x:\alpha\to d)}(A) = true$, then $V^M_\psi(E) = d$; otherwise $V^M_\psi(E) = e_\alpha$.

Let E be an expression of type α of L and A be a formula of L. Then $V^M_\psi(E)$ is the *value* of E in M with respect of ψ. it is also possible to say

that A is *valid* in M ($M \vDash A$) if $V_{\psi}^{M}(A) = true$ for all variable assignments ψ into M. A *sentence* is a closed formula of L; A is a *semantic consequence* of a set of sentences Σ ($\Sigma \vDash A$) if $M \vDash A$ for every standard model M such that $M \vDash B$ for all $B \in \Sigma$. Finally, a *theory* of simple type theory is an ordered pair $T = (L, \Gamma)$ where L is a language of simple type theory and Γ is a set of sentences called *axioms* of T; a *formula* A, therefore, is a semantic consequence of T ($T \vDash A$) if $\Gamma \vDash A$. Finally, a standard model of T is a standard model M for L such that $M \vDash B, \forall B \in \Gamma$.

Main elements on which this semantics is based are well established ideas, also used in first-order languages, such domains of individuals, truth values, models for languages, variable assignments and valuation functions defined recursively on the *syntax* of expressions. In the following section, an overview of the *audio indexing* theory will be presented.

4.2.2 Girard's System F

An interesting extension of the simple type theory is the **System F** by Girard (Girard 2006): The language is obtained by generalizing $\lambda^{\rightarrow}$ by adding the *abstraction on types*; the *types* are defined from *type variables* X, Y, Z, ... by means of two operations:

1. if U and V are types, then $U \rightarrow V$ is a type
2. if V is a type, and X a type variable, then $\prod X.V$ is a type

There are 5 schemes to create *terms*:

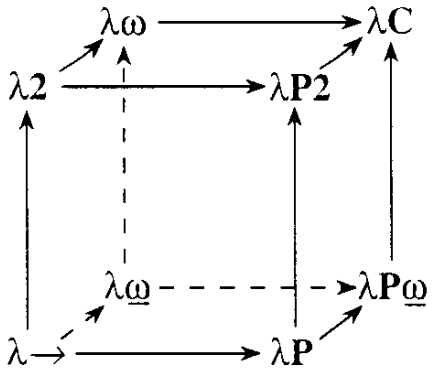
1. variables: $x^T, y^T, z^T, ...$ of type T
2. application: $t(u)$ of type V, where t is of type $U \rightarrow V$ and u is of type U
3. λ-abstraction: $\lambda x^U.v$ of type $U \rightarrow V$, where x^U is a variable of type U and v is of type V
4. universal abstraction: if v is a term of type V, the it is possible to form $\Lambda X.v$ of type $\prod X.V$ so long as the variable X is not free in the type of a free variable of v

5. universal application (exctraction): if t is a term of type $\prod X.V$ and U is a type, then $t(U)$ is a term of type $V[U \to X]$

The Λ operator represents the abstraction in the language: it does not operate on functions but on *types* of functions, promoting types to *first-class* objects. It actually operates as a second order universal quantifier; System F is then a *second order calculus with types* and is often called $\lambda 2$.

This calculus is one of the frameworks proposed by H. Barendregt (1991) to organize various calculi depicted in figure 4.3; each edge of the cube is actually the relationship of *inclusion*.

Figure 4.3: The λ-cube as proposed by Barendregdt



Most of the 8 λ-calculi are known: $\lambda^{\to}$ is the calculus with types; $\lambda 2$ is the second order polymorphic calculus (ie. System F); $\lambda\omega$ is the System Fω by Girard; λP is the AUTOMATH language for demonstrations; $\lambda P2$ has been studied by Longo and Moggi (1988); λC is the calculus of constructions introduced by Coquand and Houet (1986); $\lambda\underline{\omega}$ is related to the POLYRET system by R. de Lavalette (1985). The calculus $\lambda P\underline{\omega}$ has been studied less than the others. Each axis represent a form of abstraction:

1. types depending on terms, or *dependent types* (λP)

2. terms depending on types, or *polymorphism* (System F)

3. types depending on types, or *type operators* (System Fω)

For all the calculi of the λ-cube is valid the Church-Rosser theorem for *strong normalization*; for the Curry-Howard isomorphism they are also isomorphic to natural deduction and sequent calculus.

4.3 Basic signal models

This section will give a short overview of digital signal processing techniques (DSP) and only basic information will be provided. For a more detailed overview, there are excellent books on the topic such as (Oppenheim and Schafer 2009).

In a general sense, a discrete-time signal $\overset{n}{\vec{x}}$ is a vector of n values representing the change of a variable over time (*time series*). Most commonly, for audio signals, the variable represented is amplitude: in digital audio signals both time and amplitude are quantized in discrete quantities. The quantization of time is called *sampling rate* (f_s) and its reciprocal is called *sampling time* ($t_s = \frac{1}{f_s}$). In short, the sampling rate defines the space of a signal and determines which operations are possible.

Most of signal processing techniques assume that a given signal can be approximated by a weighted sum of functions. A signal-level representation is, therefore, a decomposition of a signal $\overset{n}{\vec{x}}$ into a linear combination of expansion functions:

$$\overset{n}{\vec{x}} = \sum_{k=1}^{K} \alpha_k \overset{n}{\vec{g}_k} \, . \tag{4.1}$$

This is a time-domain decomposition, since the signal $\overset{n}{\vec{x}}$ is a vector of n discrete values measuring amplitude over time, where K is the total number of decomposition functions. From the superposition property of linear time-invariant systems (LTI), it follows that any linear operation P applied on the signal is equivalent to the same operation applied on each single element of the decomposition:

$$P(\sum_{k=1}^{K} \alpha_k \vec[n]{g}_k) = \sum_{k=1}^{K} \alpha_k P(\vec[n]{g}_k). \tag{4.2}$$

The coefficients α_k in equation 4.2 are derived from an *analysis* stage, while the functions $\vec[n]{g}_k$ can be determined by the analysis stage or fixed beforehand and are used during a *synthesis* stage; both stages are related to a particular signal model. The choice of the decomposition functions is dependent on the particular type of application needed.

Taken together, the coefficients and the functions build the complete representation (*expansion*) of a signal: the more compact (sparse) this representation is, the more the functions are correlated to the signal (see (Goodwin 1998)).

If the functions $\vec[n]{g}_k$ used in equation 4.2 are linearly independent then they constitutes a *basis* of the signal space and the expansion is unique and mathematically invertible.

The major feature of basis expansions is that, because of their linear independence, they are very useful for certain classes of signals: some expansions, for example, provides little information about time-localized signals while they are very good for frequency-localized signals. More on this topic will be discussed in paragraph 4.3.1.

If the selection of the decomposition functions is not fixed beforhand but it is derived from an analysis stage, the decomposition is called *adaptive* . A typical way, in adaptive decompositions, to reconstruct a signal is to select from a *dictionary* of functions the ones that best match the given signal[6]. When the dictionary is made of linearly independent functions it is said to be *complete*, otherwise it is said to be *overcomplete*.

From a foundational point of view, equation 4.2 involves the following symbolic entities: a set of elements α_k, two binary operations $+$ and $\cdot$, a set of unary functions $\vec[n]{g}_k$. It is worth pointing out some important facts:

- binary operations $+$ and $\cdot$ are supposed to be commutative (i.e. $\vec[n]{g}_1 + \vec[n]{g}_2 = \vec[n]{g}_2 + \vec[n]{g}_1$ and $\alpha_1 \vec[n]{g}_1 = \vec[n]{g}_1 \alpha_1$);

[6] This is the basic principle of sparse decomposition methods such as *basis pursuit* and *matching pursuit* (Goodwin 1998).

- the set of functions $\vec{g}_k^n$ is made of elements of the same *type*, so it is an homogeneous collection.

The last point is particularly important: the representation operates on the signal in a unique way and consequently it applies only a certain *amount of knowledge* to it. Equation 4.2 belongs to a symbolic world in which only a single type of concepts is manipulated.

The discrete Fourier transform (DFT)

A typical example of signal-level decomposition is the discrete Fourier transform , given in the equation below:

$$\vec{X}_k = \sum_{i=0}^{n-1} x_i \cdot e^{-j \cdot \frac{2 \cdot \pi}{n} \cdot k}. \tag{4.3}$$

Equation 4.3 decomposes a signal into oscillatory functions represented by complex sinusoids where k is a frequency (dependent on the sampling rate f_s). In the equation above a single decomposition function k (called *channel*) is represented by $g_k = e^{-j \cdot \frac{2 \cdot \pi}{n} \cdot k}$.

The modulus $|X_k|$ and the angle $\angle X_k$ represent, respectively, the so-called magnitude and phase spectrum of the observed signal.

4.3.1 Time-frequency representations

Signal-level decompositions prefer a particular point of view on the signal. The DFT, for example, favours frequency over time because the basis functions have a definite frequency position given by $f_k = \frac{k}{T} f_s$ where f_s is the sampling rate. Time localization in DFT, however, is much poorer since it is the same for all basis functions. If the observed signal has temporally-localized event, such as a spike or any kind of quick change, the DFT will not be able to correctly represent the event that will *spread* in the transformed domain.

A possible way to practically overcome this problem is to slice a time series of N samples into small segments of n samples and separately transform each of them with the DFT, thus producing a *time-frequency representation* .

The segments, or chunks, must not be necessarily adjacent in time but they can overlap by a given amount: it is possible to take a new slice by hopping t samples from the previous slice, with $t < n$. There is an important trade-off between time and frequency localizations that will not be discussed further here; suffice it to say that time and frequency localizations strictly depend on the kind of signal analysed and must be chosen consequently.

An important time-frequency representation is the *short-time Fourier transform* (STFT) defined as a function of both time and frequency $\overset{N}{\vec{X}}_{\overset{n}{\vec{k}}}$ of a signal $\overset{N}{\vec{x}}$ of length N-samples taken n at a time while hopping by t-samples:

$$\overset{N}{\vec{X}}_{\overset{n}{\vec{k}}} = \sum_{i=0}^{N/t} \overset{n}{\vec{h}} \cdot \overset{n}{\vec{x}}_{i \cdot t} \cdot e^{-j \cdot \frac{2 \cdot \pi}{n} \cdot \overset{n}{\vec{k}}} \tag{4.4}$$

where $\overset{n}{\vec{h}}$ is a window of length n-samples (Oppenheim and Schafer 2009) and $\overset{n}{\vec{k}}$ is as above. Since the basis functions are linearly independent, the STFT is invertible; a general resynthesis equation is then given by:

$$\overset{N}{\vec{x}} = \frac{1}{n} \sum_{i=0}^{N/t} \overset{n}{\vec{X}}_{i \cdot t, \overset{n}{\vec{k}}} \cdot e^{j \cdot \frac{2 \cdot \pi}{n} \cdot \overset{n}{\vec{k}}}. \tag{4.5}$$

The matrix $| \overset{N}{\vec{X}}_{\overset{n}{\vec{k}}} |$ is called the *spectrogram* of a signal and is one of the mostly used time-frequency representations.

4.3.2 The phase-vocoder

The phase-vocoder is a very well-known technique to perform transformations on audio signals (such as time-stretching, pitch-shifting, etc.) using the STFT frequency domain representation. Since its theory is vastly documented (see (Laroche and Dolson 1999) for more information), it will be just summarized here.

The phase-vocoder operates by performing short-time Fourier transforms (STFT) on a time-domain real signal to obtain a succession of overlapped spectral frames (analysis) . The time between two spectral frames is called

hop size. The original signal can be recreated by performing an inverse Fourier transform on all frames and then adding them together (resynthesis) . Between the analysis and the resynthesis stage a number of transformations may be performed to obtain different effects on all parameters of the signal (frequency, time, etc.).

Some important improvements are possible on the phase-vocoder. First, a techinque of phase management called *phase locking* can be applied in order to improve the audio quality of the resynthesized signal. Second, a technique of amplitude management called *envelope preservation* can be applied in order to maintain the main morphology of a sound after the operation of pitch-shifting. A small review of both techniques will be given below.

Phase locking

When a signal is analysed by the DFT, each component of the signal fall in a specific channel k of the transformed domain (eq. 4.3) and has a specifc phase. Intuitively, if the component change frequency between one frame and the other, it is needed to handle its phase in order to preserve coherence in time.

One of the best approaches to preserve phase coherence in time has been proposed in (Laroche and Dolson 1999) and it is related to the estimation of the peaks in the magnitude spectrum. The basic idea is to create an entity that preserve phase coherence for each frequency analised called *phasor*. The algorithm to apply phase locking is outlined below; the steps are only intuitively described:

1. for each magnitude spectral frame $\vec{X}_i^{n\,k}$ compute the positions of the peaks (peak-map);

2. for each peak k_l in the peak-map calculate its true analysis frequency ω_a, then map this to the true synthesis frequency and synthesis phase; calculate the phasor $z_{k_l} = e^{j\phi}$;

3. for each k calculate the synthesis frame $\vec{Y}_i^{n\,k} = z_{k_l} \cdot \vec{X}_i^{n\,k}$.

Spectral envelopes preservation

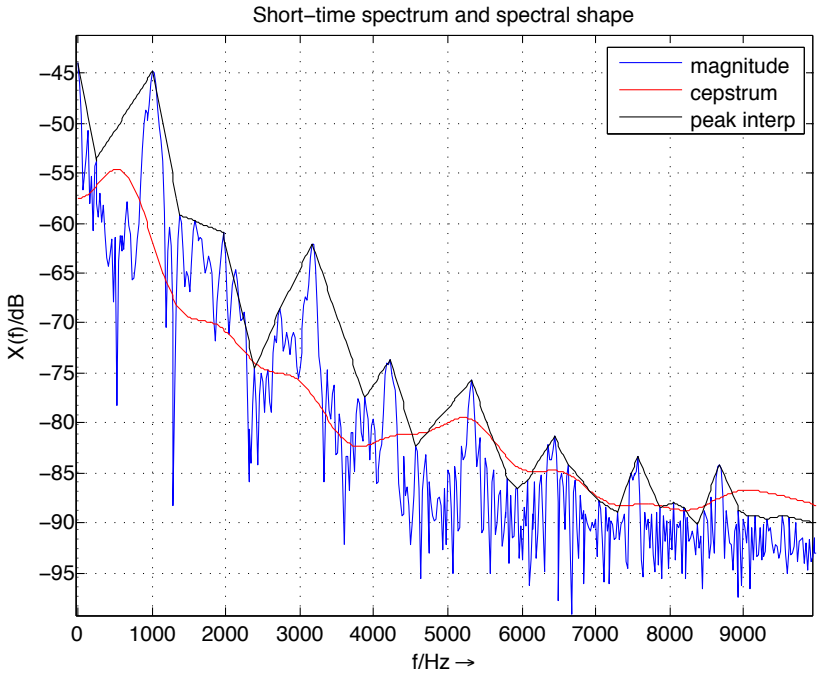


Figure 4.4: Spectral envelopes computed with the cepstrum method and with peaks interpolation.

The spectral envelope is a smooth function in the frequency-amplitude plane that matches, to a certain degree, the peaks of the amplitude spectrum. Many studies (Schwarz 1998), (Burred et al. 2006) show how spectral envelope is closely related to the of timbral information for musical signals. Spectral envelope, moreover, is quite independent of the pitch. However, by applying pitch-shifting with the phase-vocoder the spectral envelope will necessarily be transposed also. This leads to unnatural sounds that, sometimes, are really different from the original ones. To avoid this, the spectral envelope has to be kept constant, while the partials *slide* along it to their new position in frequency.

Two simple methods for envelope computation are interpolations be-

tween the peaks and the use of the *cepstrum* . The cepstrum is calculated from the discrete Fourier transform by taking the inverse transform of the magnitude of its logarithm:

$$c_p = \frac{1}{K} \sum_{k=0} log(\mid \overset{n}{\vec{X}}_k \mid) \cdot e^{j \cdot \frac{2 \cdot \pi}{K} \cdot k \cdot p} \tag{4.6}$$

where $\overset{n}{\vec{X}}_k$ is the DFT of the signal and p is the number of coefficients used in the transformation. The spectral envelope is then computed by applying a lowpass window to the cepstrum (called *liftering*) and by taking again the Fourier transform; briefly, given the signal S:

$$E = DFT(W_{LP}(c_p)) \tag{4.7}$$

where W is the lowpass window. Figure 4.4 shows a magnitude spectrum and the corresponding spectral envelopes computed by means of peaks interpolation and cepstrum. These methods present, however, several drawbacks that will not be examined here. Important extentions to the cepstral method have been proposed in literature through the concepts of *discrete cepstrum* and *true envelope*; see in (Galas and Rodet 1991) and in (Röbel and Rodet 2005) a for more information.

4.4 Clustering techniques

As pointed out in section 4.1.2, the basic idea of the approach described in this chapter is to represent music using common entities that can be instantiated into specific elements. The orthogonal way to express this concept is saying that specific elements should be *grouped* into common entities in order to have a more compact representation. The operation of grouping is related, mathematically, to the creation of classes of equivalence. In the domain of digital signals this is also related to classification (deciding the pertenance of an element to a given class) and is known with the name of *clustering* .

Clustering is the identification of groups, or clusters, of data points in a multidimensional space (usually called *feature space*). More formally, being $\{x_1, \ldots, x_N\}$ a set of N observations of a random variable x in D-dimensions,

clustering consists of partitioning the data set into a number K of groups (called *model*), with K given.

Intuitively, it is possible to define a cluster as the set of data points whose inter-point *distances* are small compared with the distances to points belonging to other clusters. Figure 4.10 depicts an hypothetical situation in which there are three evident clusters rounded by three ellipses; the radii of the ellipses represent the variance of the cluster, while the centers represent the mean. Usually, the data sets being clusterized consists of *features* describing a problem.

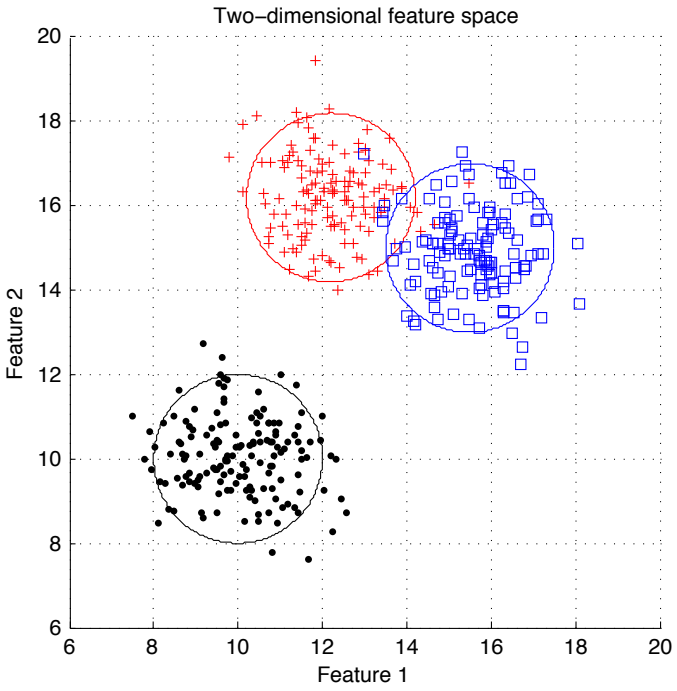


Figure 4.5: Three hypothetical clusters in a two-dimensional space; this process can be also applied to multidimensional spaces.

A possible approach to compute clusters is by using techniques such as K-means algorithm or Gaussian Mixture Models (GMM), using any kind of distances. In order to decrease the complexity of the analysis, moreover, it is

possible to *reduce* the dimensionality of the feature space by applying special techniques such as Principal Component Analysis (PCA).

To evaluate the quality of the clustering, finally, it is possible to use techniques such as the gap statistic or the Bayesian Criterion Information (BIC) that measure the *geometrical dispersion* of a given clustering model.

Next paragraphs will give a short overview of the techniques discussed above, that will be used in the theory of sound-types.

4.4.1 Low-level features for audio signals

The first important step to perform any kind of clustering is measuring a problem through specific features, in order to have an initial data set. In audio indexing, such measures are often called *low-level features* .

Low-level features are numerical values describing the contents of an audio signal according to different kinds of inspection: temporal, spectral, perceptual, etc. (Peeters April 2004) A typical example of low-level feature is the so-called *spectral shape*, represented by the statistical moments of the spectrum: *mean, variance, skewness* and *kurtosis*. The probabilistic mean over the spectrum is usually called *centroid* or *brightness* and is defined as follows:

$$\mu = \int x \cdot p(x) dx. \tag{4.8}$$

Here x are the observed data (i.e. the frequencies of the spectrum) while $p(x)$ are the probabilites to observe x (i.e. the amplitudes of the spectrum). Similarly, the variance is usually called *spread* or *bandwidth* and is defined following the previous definition:

$$\sigma^2 = \int (x - \mu)^2 \cdot p(x) dx. \tag{4.9}$$

The definition of the third and the fourth statistical moments is similar; they represent respectively the asymmetry and the flatness of the distribution.

Audio indexing

A typical field of application of clustering is *audio indexing* : by combining different techniques it is possible to group together signals that share common properties. A typical approach to audio indexing is based on the projection of some *low-level features* computed over a set of sounds in a multidimensional space; similar sounds, then, *tend* to project onto similar positions of the space, producing clusters.

4.4.2 K-means and Gaussian Mixture Models

Two of the most important clustering algorithms are K-means and Gaussian Mixture Models (GMM); both will be reviewed shortly.

Given a data set $\{x_1, \ldots, x_N\}$ of N observations the D-dimensional variable x, K-means performs the partitioning of it into k sets $\{s_1, \ldots, s_k\}$ with $k < N$, minimizing the within-cluster sum of squares J:

$$J = \arg\ \min \sum_{i=1}^{k} \sum_{x_j \in S_i} |x_j - \mu_i|^2 \tag{4.10}$$

where μ_i is the mean of the points in S_i.

A typical way to compute clusters with K-means is a through a refinement procedure called *expectation-maximization* algorithm (EM). Given an initial set of means $\{m_1, \ldots, m_k\}$, a two-fold procedure is applied:

- *expectation*: assign each element of the original data set to the cluster m_i with the closest mean

$$S_i^{(t)} = \{x_j : |x_j - m_i^{(t)}| \leq |x_j - m_{i^*}^{(t)}| \forall i^* = i, \ldots, k\}; \tag{4.11}$$

- *maximization*: calculate the new means to be the next centroid of the observations in the cluster

$$m_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j. \tag{4.12}$$

A Gaussian Mixture Model is a weighted sum of M component Gaussian densities[7]:

$$p(x|\lambda) = \sum_{i=1}^{M} w_i g(x|\mu_i, \Sigma_i) \tag{4.13}$$

where x is a D-dimensional variable, w_i are the weights for each mixture and $g(x|\mu_i, \Sigma_i)$ are the component Gaussian densities. Each component is a D-variate Gaussian function of the form:

$$g(x|\mu_i, \Sigma_i) = \frac{1}{(2\pi)^{D/2}|\Sigma_i|^{1/2}} e^{-\frac{1}{2}(x-\mu_i)'\Sigma_i^{-1}(x-\mu_i)} \tag{4.14}$$

where μ_i is the mean vector and Σ_i is the covariance matrix. As appears from above equations, the GMM is completely described by the mean vectors, the covariance matrices and the mixture weights and can be expressed in the form $\lambda = \{w_i, \mu_i, \Sigma_i\}$, with $i = 1, \ldots, M$. The GMM provides a smooth overall distribution fit of a data set and its components detail the multi-modal nature of the density. The computation of the parameters of the GMM can be also done with a two-fold procedure, aimed to maximize the *likelihood* of the model given a data set. For a sequence of training vectors $X = \{x_1, \ldots, x_T\}$, the GMM likelihood can be written as:

$$p(X|\lambda) = \prod_{t=1}^{T} p(x_t|\lambda). \tag{4.15}$$

The likelihood is a non-linear function, therefore direct maximization is not possible. The parameters of the model, however, can still be estimated using a specific variant of the expectation-maximization showed above (Bishop 2006).

Distance measures

Most of clustering methods applied on a feature space rely on the selection of a specific distance measure used to decide the pertenance of a data point to a specific cluster. Beyond Euclidean distances, other metrics are also possible

[7] For more information about Gaussian densities please see (Bishop 2006).

such as the *Manhattan* and the *Mahalanobis* distances; both will be discussed below.

The taxicab (or Manhattan) distance between two vectors in an n-dimensional real vector space with fixed Cartesian coordinate system, is the sum of the lengths of the projections of the line segment between the points onto the coordinate axes. More formally,

$$d_t(p,q) = \sum_{i=1}^{n} |p_i - q_i| \tag{4.16}$$

where p and q are vectors.

The Mahalanobis distance between two vectors in an n-dimensional real vector space with fixed Cartesian coordinate system is defined as follow:

$$d_m(p,q) = \sqrt{\sum_{i=1}^{n} \frac{(p_i - q_i)^2}{\sigma_i^2}} \tag{4.17}$$

where p and q are vectors and σ_2^i is the standard deviation. This formulation is often called *normalized euclidean distance*.

4.4.3 Dimensionality reduction

The clustering methods discussed above can be applied on variables of any number of dimensions. In some circumstances, however, can be useful to reduce the number of dimensions used in the analysis.

Dimensionality reduction can be applied by means of two main approaches: feature selection and feature extraction.

Generically, feature selection tries to find a subset of n features from the original set of D features with $n < D$ by minimizing or maximizing some cost functions. In feature extraction, on the other hand, the original set of D features is transformed into a new set with less dimensions. Among the most important techniques for dimensionality reduction by feature extraction there is principal compoment analysis (PCA) .

PCA (also called *Karhunen-Loéve transform*) is a linear and orthogoanl transform that looks for a number of uncorrelated variables (called princi-

pal components) from a set of correlated variable, without assuming any particular distribution property. It is defined as an orthogonal linear transformation that transforms the data to a new coordinate system such that the greatest variance by any projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on. Figure 4.6 shows an example application of PCA.

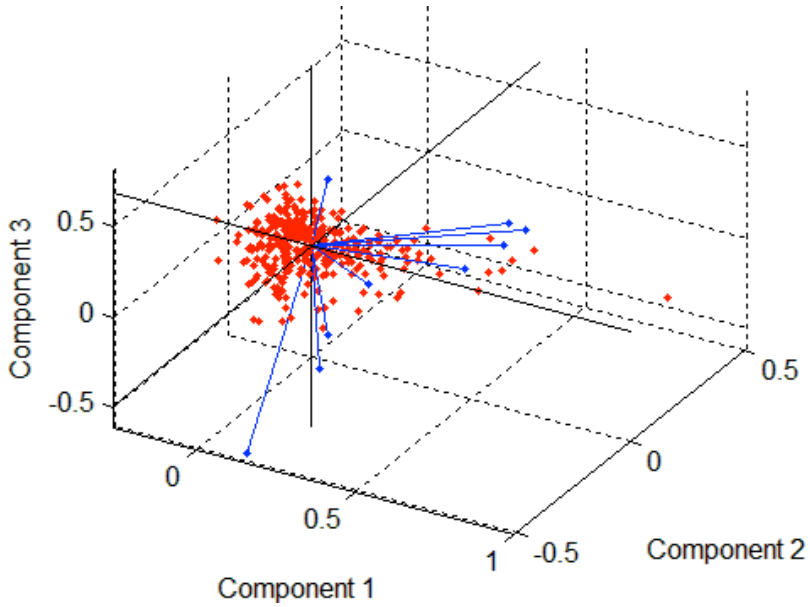


Figure 4.6: Principal compoments computed on a set of correlated variables.

Briefly, for a data matrix X^T with zero empirical mean, where each row represents a different repetition of the experiment, and each column gives the results from a particular probe, the PCA transformation is given by:

$$Y^T = X^T W = V\Sigma^T \tag{4.18}$$

where the matrix Σ is an m-by-n diagonal matrix with nonnegative real numbers on the diagonal and $W\Sigma V^T$ is the singular value decomposition (SVD) of X. See (Bishop 2006) and (Press et al. 2007) for more information.

4.4.4 Model evaluation

With clustering, a particular problem is described by partinioning it into K subclasses that should represent it adequately. Anyway, it is not easy to determine if a particular model is a *good* representation of a problem. There exist methods, however, to measure the relative goodness of fit of a statistical model. Such methods can be used to compare how well different clusterings perform on a given data set. Usually, two main approaches are used to evaluate a model:

- *internal criterion of quality*: it is possible to assign a score to an algorithm by analysing the relations between the produced clusters, where models with high similarity within a cluster and low similarity between clusters will get a good scoring; high scores, however, not always result in effective information retrieval applications;
- *external criterion of quality*: the score is assigned to a model by comparing the results of the clustering against some external benchmark (pre-classified items) often created by humans (experts).

In the first category there are two important quality measures for a model: gap statistic and Bayesian information criterion (BIC).

The gap statistic is an error measure for clustering that looks at the within-cluster dispersion $W_k = \sum_{l=1} k \frac{1}{2n_l} D_l$, where D_l is the pairwise squared distance of all points in a given cluster. Formally:

$$d_m(p,q) = (1/B) \sum_{b=1}^{B} log(W_k^*) - log(W_k) \tag{4.19}$$

where B is a reference data set in the range of the observed data and W_k^* is its within-cluster dispersion.

The Bayesian information criterion (BIC) is a criterion for model selection among a class of parametric models with different numbers of parameters defined as:

$$BIC = -2 \cdot ln(L) + k \cdot ln(n) \tag{4.20}$$

where L is the maximum likelihood for the model, k is the number of clusters and n is the number of data points.

Automatic estimation of K

By assessing a quality measure for models, it is also possible to automatically select a model for a given problem. A simple approach is to iteratively change the number K of clusters and compute a quality measure for each K. it is then possible to study the function of quality measures to select the best model.

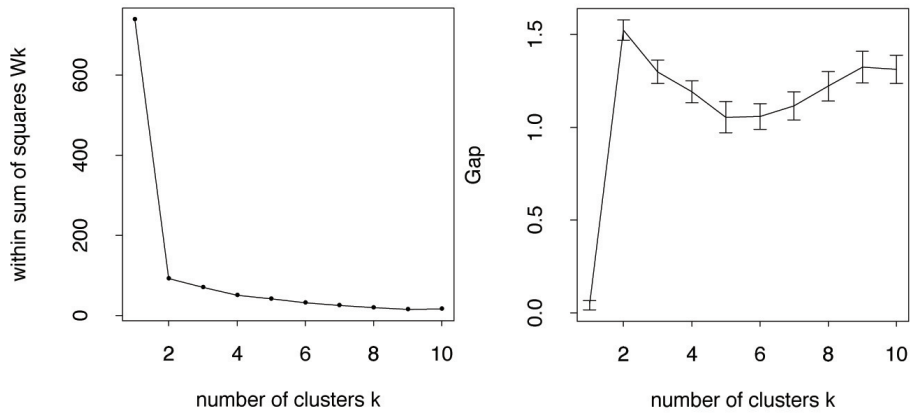


Figure 4.7: A plot of the variation of the within-cluster dispersion changing the number of clusters.

In many situations an automatical estimation of the best fitting model is also feasible. As an example, figure 4.7 represents a model evaluation based on the gap statistic. The left plot depicts the change of the within-cluster dispersion depending on K; the right plot, instead, the corresponding value of the gap statistic. The fast drop on the left plot (that represents the best fitting model) corresponds, in the right plot, to the maximum of the gap statistic. Taking the maximum of the gap statistic, then, can be an approach to automatic selection of K.

4.4.5 Markov models

The clustering analysis described above can be very useful in describing problems in a *static* way. Some problems, however, happens in time and have a *dynamic* nature. Markov models , while not directly connected with clusterings, are useful tools to describe *sequential properties* of a problem.

A *Markov model* is a stochastic model represented by a directed graph that can have infinite loops. The edges of the graph are labelled with *transition probabilities* such that the sum of outgoing probabilities from a single node is 1. A realization of a Markov model is a random path that moves from state to state according to model's probabilites. These can be represented as a *transition matrix* T in which each element T_{ij} is the probability of moving from state i to state j. A valid transition matrix must satisfy the following properties: $0 \leq T_{ij} \leq 1$ and $\sum_j T_{ij} = 1$.

Table 4.1: Transition probabilites for principal harmonic families.

	Tonic	Subdominant	Dominant
Tonic	.25	.5	.35
Subdominant	.35	.25	.5
Dominant	.5	.35	.25

An example will clarify the exposed ideas: table 4.1 represent an hypotetical transition table for the principal functions in tonal harmony; the corresponding graph is depicted in figure 4.8.

In the so-called *hidden Markov model* (HMM) there is no direct observation of the state of the model. From a given state i of a total of M states, the model emits a symbol k probabilistically chosen from a set of K symbols, whose probability is denoted by:

$$h_i(k) = P(k|i) \tag{4.21}$$

with $(0 \leq i < M, 0 \leq k < K)$. More on HMM can be found in (Rabiner February 1989).

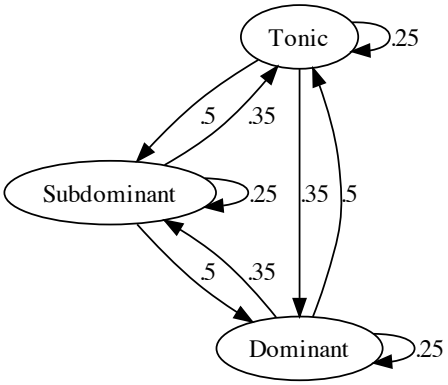


Figure 4.8: A graph representing transition probabilities of table 4.1.

4.5 Sound-types

After the presentation of the needed theoretical background, given in the above sections, it is now possible to formulate the central argument of this chapter: the *theory of sound-types*.

The *theory of sound-types* is a new representation method for musical signals that, while being generic enough to be used for different signals, fulfills *by-design* the following requirements:

- **signal-dependent semantics**: the basis of the representation are inferred from the signal, using learning techniques; this creates the possibility to describe concepts that are really *related* to the sound being analysed (adaptive dictionary);

- **scalability**: it is possible to change the *degree of abstraction* in the representation, ranging from the signal level to the symbolic- level in a *continuous* manner; the degree of abstraction becomes a parameter of the representation;

- **weak invertibility**: the representation method is able to generate the

represented signal; this possibility does not imply, however, that the generated signal must be waveform-identical to the original one, but only that *perceptually relevant* parts of it can be reconstructed (that's why it is called weak);

- **generativity**: it is possibile to generate sounds *other* than the original one, according to some parameters in the domain of the representation that can be estimated from a given signal or deliberately created.

4.5.1 The *typed* model

The basic idea of sound-types is to represent sounds by means of *types* and *rules* inferred by some low-level descriptions of signals (Peeters April 2004) and subsequent learning stages. The types represent **classes of equivalences** for sounds, while the rules represent **transition probabilities** that a type is followed by another type.

This means, mathematically, to *translate* a signal-level representation into other forms involving different elements and operators (mid to symbolic-level representations); more formally:

$$\begin{aligned} x[n] &= \textstyle\sum_{k=1}^{K} \alpha_k g_k[n] \\ &= \alpha_1 g_1[n] + \ldots + \alpha_k g_k[n] \\ &= \beta_1 f_1[n] + \ldots + \beta_j f_j[n] \\ &\vdots \\ &= \omega_1 h_1[n] + \ldots + \omega_t h_t[n]. \end{aligned}$$

In the equations above $\alpha, \beta, \ldots, \omega$ could be any kind of weighting coefficients, $g_k, f_j, \ldots, h_t$ are variables belonging to different *types*, $+$ and $\cdot$ are relations defined for each type and $t < j < \ldots < k$ (i.e. last equation has less elements than first equation). Notice that $+$ and $\cdot$ are not algebraical sum and multiplication and are *not* required to be commutative: they can be any kind of binary relation defined over specific types. As long as it is possible to

convert from type g_k to type h_t and to define relations on both, it is possible to perform the translation. Since the $+$ relation is not the algebraical sum, it is possibile to suppose that a symbolic-level representation is a *sequence* of types and that $+$ is the *successor* function (i.e. $g + f$ means that variable f of type F follows variable g of type G[8]; remember that if F and G are types then $G \rightarrow_+ F$ is a type).

4.5.2 The sound-types transform

From a purely theoretical standpoint, all the theory presented above is based on a particular equation called *sound-types transform*. This section will define such a transform, will examine some of its properties and will relate it to the short-time Fourier Transform. In paragraph 4.5.4 it will be shown how this transform is used in the whole process.

Given a signal $\overset{N}{\vec{x}}$ of length N-samples and a window $\overset{n}{\vec{h}}$ of length n-samples, it is possible to define an **atom** as a windowed chunk of the signal of length n-samples (the starting position of the chunk is not indicated here):

$$\overset{n}{\vec{a}} = \overset{n}{\vec{h}} \cdot \overset{n}{\vec{x}} \tag{4.22}$$

where the operator $\cdot$ is a multiplication *element-by-element*. Using an adequate hop-size t during the analysis stage (for example $t \leq n/4$), it is possible to reconstruct a *perfect*[9] version $\overset{N}{\vec{x}'}$ of the original signal with a sum of atoms as a function of time[10]:

$$\overset{N}{\vec{x}'} = \sum_{i=0}^{N/t} \overset{n}{\vec{a}}_{i \cdot t} \tag{4.23}$$

where N/t is the total number of atoms present in the signal $\overset{N}{\vec{x}}$. It is possible, after the computation of a set of low-level features on each atom of $\overset{n}{\vec{a}}_i$, to

[8] The *successor* relation is evidently non-commutative.

[9] As in STFT, the reconstruction can be perfect only under special conditions (not detailed here) deriving from the type of window used and from the overlapping factor.

[10] The positions in time of the blocks of n-samples are given by an index i that counts the number of hops (ie. $i = 4 \implies 4 \cdot t$).

define a **sound-cluster** as a set of atoms that *lie* in a defined area of the feature space (ie. that share a *similar* set of features):

$$\overset{k_r}{\vec{c}_r} = \{\overset{n}{\vec{a}}_{r,1}, \ldots, \overset{n}{\vec{a}}_{r,k_r}\}. \tag{4.24}$$

The content of $\overset{k_r}{\vec{c}_r}$ is given by a statistical analysis applied on the feature space that decides the position of each sound-cluster and its belonging atoms.

A **model** $\mathcal{M}_{\overset{N}{\vec{x}}}$ of the signal $\overset{N}{\vec{x}}$ is the defined as the set of the clusters discovered on it:

$$\mathcal{M}_{\overset{N}{\vec{x}}} = \{\overset{k_1}{\vec{c}_1}, \ldots, \overset{k_r}{\vec{c}_r}\}. \tag{4.25}$$

The cardinality $|\mathcal{M}_{\overset{N}{\vec{x}}}|$ of the model is also called the **abstraction level** of the analisys; since the number atoms is N/t it is evident that $1 \leq |\mathcal{M}_{\overset{N}{\vec{x}}}| \leq N/t$ with higher abstraction being 1 and lower abstraction being N/t.

Each sound-cluster in the feature space has an associate **sound-type** $\overset{n}{\vec{\tau}_r}$ in the signal space, defined as the weighted sum of all the atoms in the sound-cluster where the weights $\overset{k_r}{\vec{\omega}_r}$ are the distances (any kind of Bregman's divergences) of each atom to the center of the cluster:

$$\overset{n}{\vec{\tau}_r} = \sum_{j=1}^{k_r} \overset{n}{\vec{a}}_{r,j} \cdot \omega_{r,j} \tag{4.26}$$

with $\omega_{r,j} \in \overset{k_r}{\vec{\omega}_r}$. The whole set of sound-types in the signal $\overset{N}{\vec{x}}$ is called **dictionary** and is the equivalent, in the signal-space, of the model in the feature-space:

$$\mathcal{D}_{\overset{N}{\vec{x}}} = \{\overset{n}{\vec{\tau}_1}, \ldots, \overset{n}{\vec{\tau}_r}\}. \tag{4.27}$$

The creation of a sound-type from a sound-cluster is also called *collapsing* and can be indicated with the symbol $\langle \overset{k_r}{\vec{c}_r} \rangle = \overset{n}{\vec{\tau}_r}$: this operation represents an interesting connection between the feature space and the signal space that leads to the equivalence $\langle \mathcal{M}_{\overset{N}{\vec{x}}} \rangle = \mathcal{D}_{\overset{N}{\vec{x}}}$.

It is possible to define a function Ψ that maps an atom to its corresponding sound-type as:

$$\Psi_{\substack{n \\ \vec{a}_i}} : \overset{n}{\vec{a}_i} \rightarrow \langle \overset{k_r}{\vec{c}_r} \rangle. \tag{4.28}$$

For a complete decomposition of the signal, it is also useful to define a function Θ that returns the original time position of each atom:

$$\Theta_{\substack{n \\ \vec{a}_i}} : \overset{n}{\vec{a}_i} \rightarrow i. \tag{4.29}$$

It is now possible to define the **sound-types decomposition** $\overset{N}{\vec{x}}''$ of a signal by *replacing* each atom of equation 4.23 with the corresponding sound-type defined through Ψ, in the right time position given by Θ:

$$\overset{N}{\vec{x}}'' = \sum_{i=0}^{N/t} \overset{n}{\vec{\tau}}_{r,p} \tag{4.30}$$

where $p = \Theta_{\substack{n \\ \vec{a}_i}}$. Finally, it is possible to define a function of time and frequency by multiplying the sound-types in a given dictionary with complex sinusoids:

$$\boxed{\overset{N}{\vec{\Phi}}_{\substack{n \\ \vec{k}}} = \sum_{i=0}^{N/t} \overset{n}{\vec{\tau}}_{r,p} \cdot e^{-j \cdot \frac{2 \cdot \pi}{n} \cdot \overset{n}{\vec{k}}}} \tag{4.31}$$

where $\overset{n}{\vec{k}} = \{ f_1, \ldots, f_n \}$ is a vector of frequencies. Equation 4.31 is called the forward **sound-types transform** (STT) ; the inverse transform can recreate the sound-types decomposition and is given by:

$$\overset{N}{\vec{x}}'' = \frac{1}{n} \sum_{i=0}^{N/t} \overset{n}{\vec{\Phi}}_{\substack{n \\ i \cdot t, \vec{k}}} \cdot e^{j \cdot \frac{2 \cdot \pi}{n} \cdot \overset{n}{\vec{k}}}. \tag{4.32}$$

As the next section will show, equation 4.31 is connected to STFT.

4.5.3 STT and STFT

Equations 4.31 and 4.4 have a strong resemblance. As observed in the previous section, the abstraction level of a model can be at most equal to the number of atoms (N/t) in the original signal. The extreme case for $|\mathcal{M}| = N/t$ is interesting: for that abstraction level, each sound-cluster is a singleton made of a single atom and consequently each sound-type reduces to that single atom scaled in amplitude:

$$|\mathcal{M}| = N/t \implies \vec{c}_r = \{\vec{a}_1\} \implies \vec{\tau}_r = \vec{a}_r \cdot \omega_{r,1}. \tag{4.33}$$

For equation 4.22, an atom is defined simply a windowed chunk of the original signal. Not considering the amplitude scaling factor, this also makes the sound-types decomposition $\vec{x}''$ equivalent to the simple decomposition $\vec{x}'$, leading to the important consequence that STT is a **generalization** of STFT:

$$\boxed{\vec{\tau}_r = \vec{a}_r = \vec{h} \cdot \vec{x} \implies \sum_{i=0}^{N/t} \vec{\tau}_{r,p} \cdot e^{-j \cdot \frac{2 \cdot \pi}{n} \cdot \vec{k}} = \sum_{i=0}^{N/t} \vec{h} \cdot \vec{x}_{i \cdot t} \cdot e^{-j \cdot \frac{2 \cdot \pi}{n} \cdot \vec{k}}} \tag{4.34}$$

with p defined as above. This property also holds for the inverse transform case but the prove will not be given here. The abstraction level of a model is directly connected to the *goodness* of the representation: the higher the abstraction (closer to 1) the more compact the representation. On the contrary, the quality of the synthesis given by the inverse transform degrades with high abstractions and increases with low abstractions becoming a perfect reconstruction for $|\mathcal{M}| = N/t$ as proved above.

4.5.4 The computation of sound-types

In order to provide a verifiable model for the proposed theory, some functions are needed to clearly define whether a given variable belongs to a given type and how it is possible to convert from a type to another. A possible way to achieve these requirements is through a two-fold process divided into the following stages:

- **types inference**: during this stage the types involved in the representations are discovered;

- **rules inference**: a second stage is needed to discover the relations between the types.

This is an iterative process and must be repeated until there are no more rules to discover; this will be cleared later on.

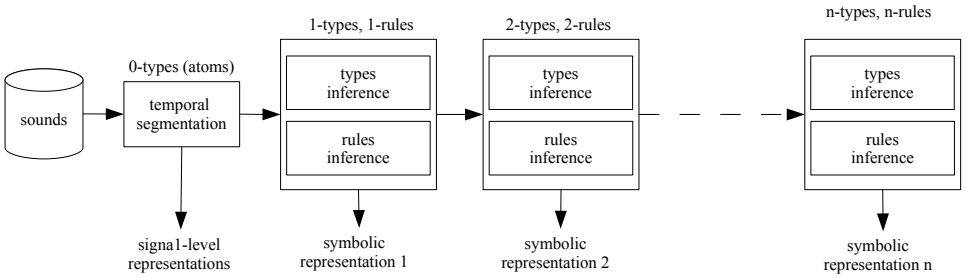


Figure 4.9: An outline of the proposed algorithm for types and rules inference

The analysis stage

The following procedure shows a possible realization of the two-fold process, using low-level descriptors plus statistical learning for types inference and Markov models for rules inference; the first step of the algorithms is represented by temporal segmentation:

1. **atoms creation**: subdivide a sound into small chunks of approximately 40 ms called *atoms* or *0-types* overlapping in time and frequency (labelled them with integer numbers); these atoms can be produced either by simply overlapped windows, by onsets separations or by other approaches such as atomic decomposition;

2. **1-types inference**: compute a set of low-level descriptors on each atom obtained in the previous step, project the descriptors in a multi-dimensional space and compute the *clusters* by means of statistical techniques; each cluster will represent a *1-type* (let's label them $g_1, f_1, \ldots$);

3. **1-rules inference**: implement a Markov model to describe the sequences of types present in the analysed sound (*1-rules)*;

4. **1-level representation**: represent the sound in a symbolic language using the discovered 1-types and 1-rules and create sequences of types depending on the rules;

5. **n-types inference**: compute a set of low-level descriptors on the whole sequences found in previous steps (for example $g_1 + f_1$); project again the descriptors and compute again the clusters: each cluster will represent a *n-type* (let's label them $g_n, f_n, \ldots$);

6. **n-rules inference**: implement a Markov model to describe the sequences of types present in the analysed sound (*n-rules)*;

7. **n-level representation**: represent the sound in a symbolic language using the discovered n-types and n-rules and create sequences of types depending on the rules;

8. **repeat n-rules and r-types**: until valid rules are found.

Algorithm 2 details the described procedure in pseudo-code.

The number of iterations of the whole process are called the *abstraction levels* of the representation[11]. In terms of atomic decomposition, all the sets of the discovered types are time-frequency atoms with different time scales and spectral content; the higher the level of a type the less it is generic, the more expressive. Figure 4.9 illustrates the proposed approach. Low-level descriptors and statistical techniques are not used to classify different sounds, but *parts* of a single sound; another approach could take into account a real population of sounds and compute sound-types over a whole database; since different atoms and sequences (moleculae) belong to the same type as long as

[11] For a formal definition of abstraction see paragraph 4.5.2

Algorithm 1 Sound-types analysis

Require: signal s
  decompose s in atoms $a[n]$
  repeat
    for every atom in $a[n]$ **do**
      compute m-dimensional feature space $f_{n,m}$
    end for
    compute optimal number of clusters k
    compute clusters $c[k]$ on $f_{n,m}$
    synthesize k types from $c[k]$
    create a representation r of s using clusters $c[k]$
    for every cluster in $c[k]$ **do**
      compute transition probabilities $p_{k,k}$
    end for
    synthesize sequences of types with non-null probability
    $a[n] \Leftarrow$ synthesized sequences of types
    $n \Leftarrow k$
  until no more transitions

they share common properties (defined by the set of descriptors), they could theoretically be shared between different sounds. From an acoustical point of view, the information amount increases dramatically from level to level, ranging from the so-called *acoustical quanta* to segments of sounds that could be even recognized as *sections* of a musical composition. The representaion created on each level can be done on a symbolic language of choice, even with simple strings of labels.

The synthesis stage

The synthesis of the discovered types is a relatively easy task and can be done either in time of frequency.

In time it is basically the application of the sound-types decomposition, a weighted sum of all the atoms belonging to the same cluster, in which the distance of the cluster is the weight. In frequency, on the other hand, is done

through the application of the sound-types transform. While the both STT and sound-types decomposition are formally defined using weighed sums, other methods are also possible. For example, a *witness* of a type can be selected from the cluster (either randomly or by its distance to the center of the cluster).

Algorithm 2 details the synthesis procedure in pseudo-code, for the case of frequecny domain reconstruction.

Algorithm 2 Sound-types synthesis

Require: n-level representation r
Require: dictionary of n-types $a[n]$
  for every symbol in r **do**
    compute the forward sound-types transform on $a[n]$
    apply needed transformations on sound-types
    compute the inverse sound-types transform
    overlap-add corresponding type
  end for

The overall quality of the reconstructed signal strictly depends on the number of types used and on the synthesis method selected. More on this problems will be discussed in section 4.6.

4.5.5 The link with the phase-vocoder

The whole mechanism for sound-types analysis and synthesis described above is really close to the one performed by the phase-vocoder , described in paragrah 4.3.2. The big difference between the two approaches is that in the former the sound-types transform is used instead of the short-time Fourier transform. Since in paragraph 4.5.3 has been proved that STT is a general case of STFT, it is possible to adfirm that the theory of sound-types is a sort of extension of the phase-vocoder. For this reason, it also configures as a full framework for sound analysis and synthesis.

4.6 Current status

At the moment of the writing of this work, the available implementation does not cover all the parts of the analysis-synthesis algorithm for sound-types analysis. A partial implementation can compute, however, low-level features, clusters in the feature space and transition probabilies up to the first level.

The analysis-synthesis framework is perfectly functional but the symbolic representation is only possibile with 1-types and 1-rules thus proving the theory of sound-types only partially.

A typical types-inference stage (performed by clustering) is represented in figure 4.10: common sound-atoms are grouped in the same cluster and relevant elements of the clusters (such as centroid, spread, etc.) are computed.

A rules-inference stage (performed by a Markov model), is depicted in figure 4.11: the nodes are the sound-types, while the connections are the transitions between them.

With the two stages computed for the first level (1-types, 1-rules), it is possible to represent a signal in a pseudo-symbolic way through a *string of labels*. After the analysis stage, a dictionary of the found types (basically sound grains created as described in section 4.5.4) and a simple string are produced: each label at position k in the string represents the corresponding type (through a numeric index that refers to the position in the dictionary). In general, the algorithm creates a *compact* representation of the given sound; the size of the representation is directly connected to the number of types discovered. If a sound is represented with 33% of sound-types (i.e. a type each three atoms) the compression ratio will be roughly 60%.

Implemented features

The complete list of implemented features in the current version is the following:

- **low-level features**: spectral centroid, spectral spread, spectral skewness, spectral kurtosis, spectral irregularity, specrtal slope, spectral de-

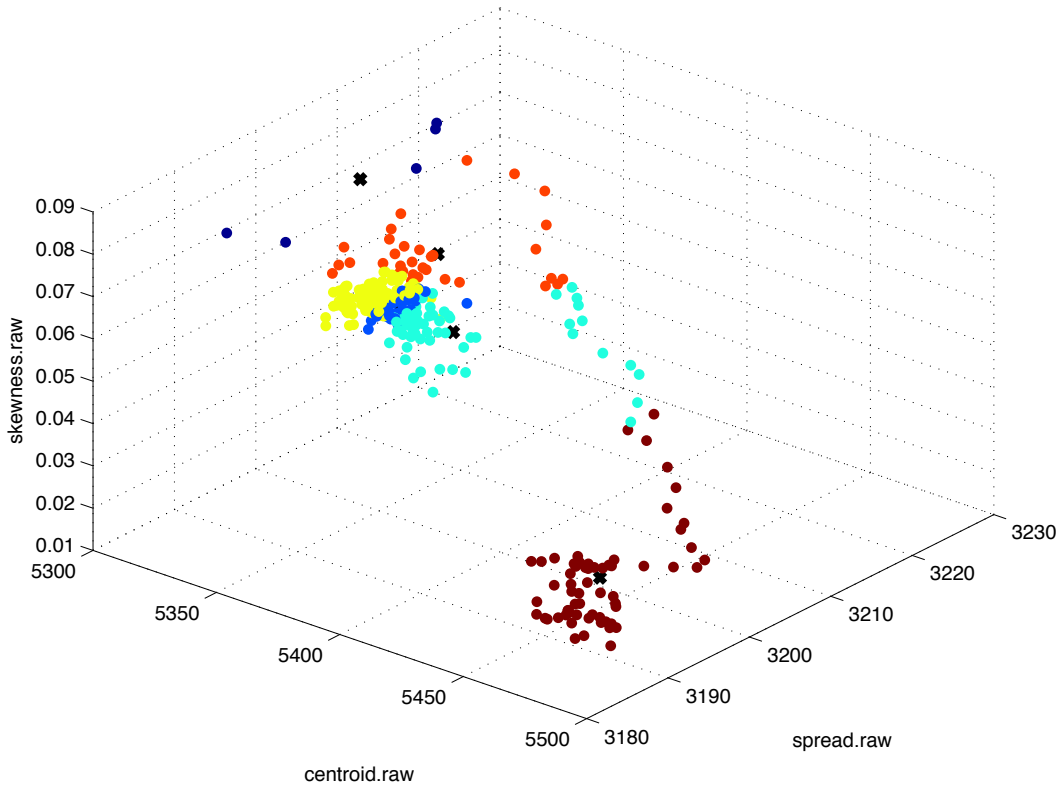


Figure 4.10: A typical clustering stage in the feature space; here the sound has been analysed with spectral centroid, spectral spread and sectral skewness.

crease, high frequency content, spectral flux, energy, zero-crossing rate, fundamental frequency, inharmonicity;

- **clustering algorithms**: K-means, gaussian mixture models (GMM);
- **auto-estimation of number of clusters**: it is possible to automatically estimate the optimal number of clusters by means of two distinct techniques for each clustering algorithm (gap statistic for K-means and BIC measure for GMM);
- **dimensionality reduction**: by means of principal component analysis

(PCA) it is possible to compute as many features as wanted and then reduce the analysis to a smaller number of dimensions;

- **transition probabilities**: Markov models;
- **resynthesis algorithm**: the resynthesis algorithm works both in frequency and in time domain with types interpolation. In the reconstructed signal, where a long sequence of the same type is found, it is possible to create linear interpolation with next appearing type, improving sound quality;
- **distance measures for resynthesis**: euclidean, Manhattan distance (taxicab), Mahalanobis distance, cosine similarity;
- **symbolic representations**: strings of labels.

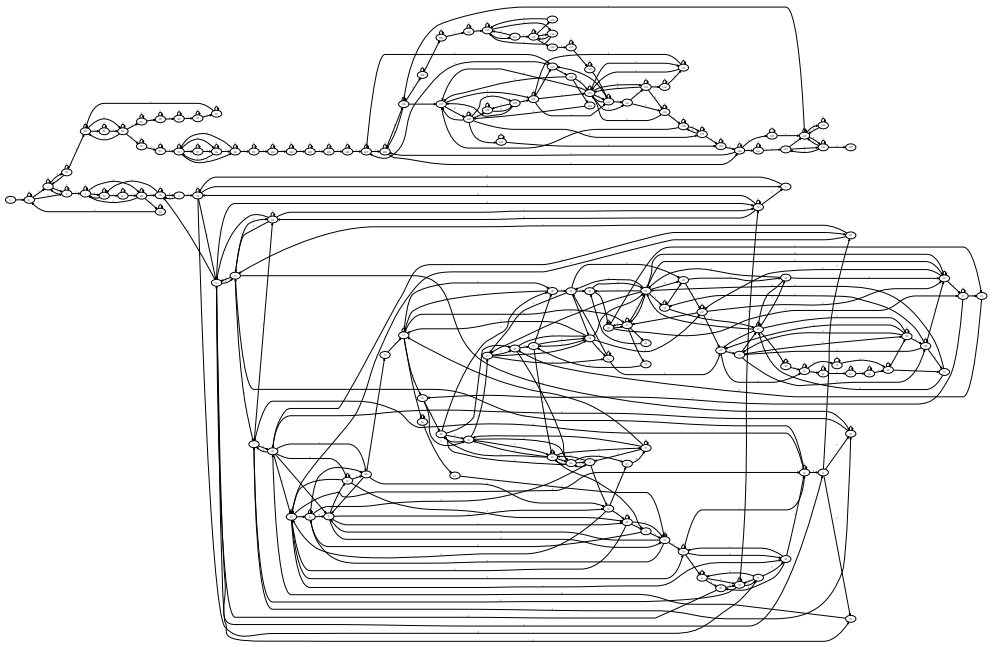


Figure 4.11: Transitions probabilities for the first level.

4.7 Applications

Several applications of the algorithm are currently possible on musical signals. Since the theory is still under development, however, not all the possibilities have been thoroughly tested. Nontheless, to show the potential of sound-types, a short list of experimented applications is given below.

Audio compression

While not being the main purpose of the approach, it is possible to apply a lossy compression to a sound by a given ratio[12]. This possibility comes from the fact that a signal is represented with few sound-types instead of many atoms.

The quality of compression, anyway, is not comparable with dedicated algorithms such as MP3. Intuitively, increasing the compression ratio produces a sort *dispersion* effect on the signal.

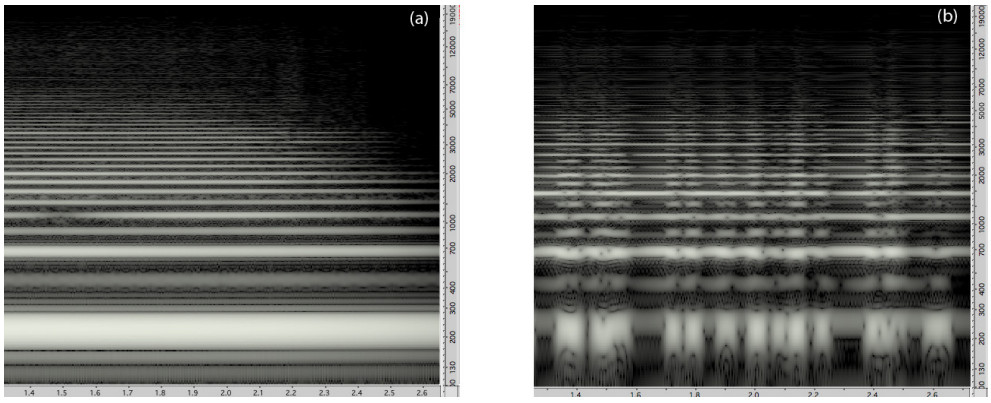


Figure 4.12: A comparison of the spectrogram between an original sound *(a)* and it is compressed version *(b)*.

Figure 4.12 shows a comparison between an uncompressed sound (a) and it is compressed version (b). The compression ratio is about 90% and in plot (b) it is possible to see how the reconstruction of the partials is a bit *fuzzy*.

[12]For more information about lossy and loseless compressions see (Press et al. 2007).

This is a clear example of bad reconstruction: after some experiments and consequent acoustic inspection it has been possible to determine the ratio $1 : 10$ as the maximum compression value for preserving the intellegibility of the audio signal.

More formalized experiments, however, should be done in order to effectively evaluate the compression capabilities of the theory of sound-types.

Time and frequency transformations

Since the theory of sound-types is an extension of the phase-vocoder, it is possible to perform various transformations such as time-stretch, denoising and pitch-shift; for the latter, formant preservation by means of cepstral envelope and phase locking have been also implemented. Other exotic effects are also possible, such as *robotization*.

Some samples processed by the proposed method can be found online at *http://www.soundtypes.com*.

Probabilistic generation

Using the discovered probabilities and types it is possible generated sounds *affine* to the original ones, by means of a biased random generator; this corresponds to a realization of the Markov model created during the analysis.

This approach has been tested on a small jazz corpus (including several instruments) to imitate the style of improvisations. While no extensive testing has been done, the results on the jazz corpus are promising: some instrumental solos (for example on double-bass) are exceptionally well imitated.

Quasi-symbolic description

By analysing the created string of labels, it is possible to acquire information of *salient* properties of the sound and represent such information in a meaningful way. Figure 4.13, for example, shows a comparison between a circular graph created with collected types and probabilities (using the same approach as figure 4.11) and a typical structure representation from the soft-

ware *OpenMusic*[13].

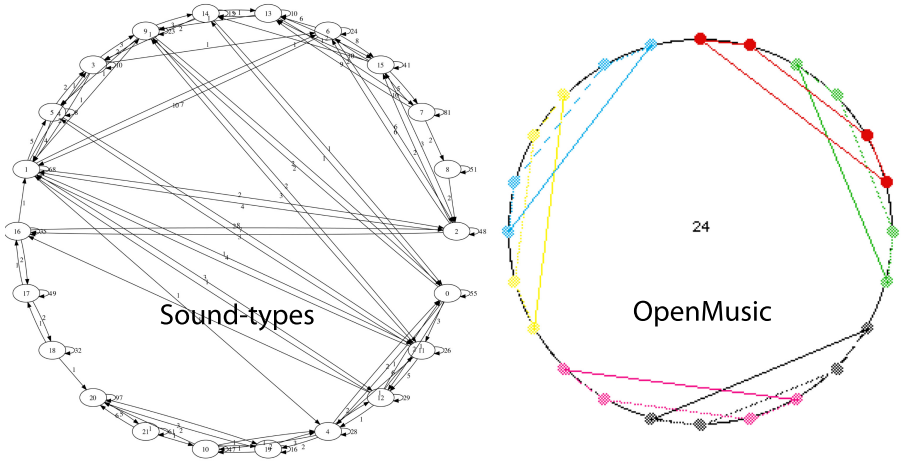


Figure 4.13: A comparison between sound-types and OpenMusic graphs

The integration of the purely symbolic world of OpenMusic and of the quasi-symbolic approach of sound-types could be very interesting for musical representations.

4.8 Open problems

Since the theory discussed above is in an early development stage, many problems are still open. Two, among the others, are really imporant: the *reduction effect* and the lack of evaluation procedures.

Reduction effect

The reduction effect is not a real problem of the theory, since it can be considered as a feature. Shortly, the more the number of clusters reduces (meaning that more entities are grouped in the same sound-type) the better will be the

[13]This is software is a well known tool developed at IRCAM to perform computer-aided composition; for more information see chapter 3. It can be found at *http://repmus.ircam.fr/openmusic/home*.

representation but on the same time, however, the worst will be the sound resynthesis. Sound quality is directly linked to the number of clusters: increasing the latter produce an improvement of the former. Nontheless, augmenting number of clusters reduces the possibility of having many levels in the analysis. No easy solutions have been found at the moment for this effect.

Evaluation procedures

It is not easy fo find evaluation procedures for the proposed method; since there are several possible applications, each of them should be carefully tested. The only technique adopted for testing is, at this stage, acoustic inspection of reconstructed signals after full analysis and synthesis. Sound transformations, audio compression and probabilistic generation can all be evaluated by comparing the results with the expectations. However, more scientific procedure to evaluate *sound quality* should be applied.

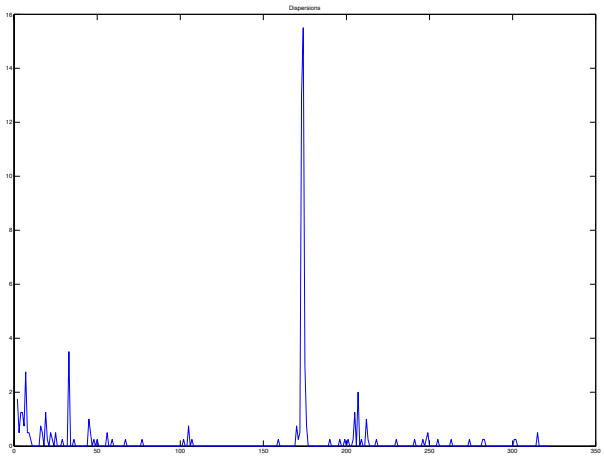


Figure 4.14: A plot of clusters dispersions as a quality measure of the method; the highest peaks signify bad clusters.

A possible measure of the *clustering quality* could be created by means of the gap statistic. Using that technique, in fact, it is possible to evaluate bad clustering looking at dispersions: in figure 4.14 the highest peaks signify

bad clusters. Anyway, this quality measure is hardly related to the quality of the reconstructed signal and a good evaluation procedure is still under investigation.

4.9 Summary

In this chapter, the theory of sound-types has been presented. A review of the main conceptual tools used by the theory has also been given and both the logical and computational backgrounds have been discussed.

The theory takes inspiration from the simple type theory and from the System F by Girard. The central idea is to represent sound and music using *types* and *rules*. To effectively compute these entities, many signal processing techniques are used.

After a presentation of basic signal models, a review of clustering methods is given. The main topics discussed are: low-level features for audio description, K-means and GMM, dimensionality reduction through PCA and model evaluation with gap statistic and BIC measure.

The theory is based on the sound-types transform (STT). The relation between STT and SFTF is also given, showing that the former is a general case of the latter.

At the end of the chapter, a concrete implementation of the theory of sound-types is showed and the list of possible applications is given. Since the theory of sound-types can be thought as an extension of the phase-vocoder, many sound trasformations are available. While extensive testing of the method has not been done yet, among possible applications there are: audio compression, phase-vocoder effects, probabilistic generation and quasi-symbolic representation of audio signals. All these applications are experimental and could be interesting, in future developments, to select and expand only some of them.

Chapter 5

Conclusions and perspectives

Why do rhythms and melodies, which are composed of sound, resemble the feelings, while this is not the case for tastes, colors or smells? Can it be because they are motions, as actions are also motions?

Aristotle, Prob. xix. 29

Abstract

This chapter will summarize main results of this research and will outline some possible expansions of the proposed theory.

5.1 From theoretical to computational models

Music has always been linked to symbols. Any musical score is, essentially, a set of symbols that musicians decode and interpret. This important connection is probably related to the dual nature that music manifests.

On one side, it is purely emotional and escapes any formalization. On the other side, however, music is intimately connected to mathematics and its interpretation as an art is relatively recent.

This work tried to investigate principal attempts done in formalizing music with mathematical languages of various kinds. All these methods have, probably, a common root in Riemann's interpretation of music. In his work *Musicalische Logik: Ein Beitrag zur Theorie der Musik* the expression *musical logic* appears for the first time. This new point of view originated two main paths: one oriented to music representation by means of symbolic logic, the other more focused on algebraic methods. Both applied a symbolic analysis

to objects that were, intrinsecally, *already* symbolic. Nonetheless, their fortune has been very different.

The former did not have much luck. After first developments at the beginning of last century, it stopped for more than fifty years. New developments started again in the seventies but, unfortunately, not so many researchers and composers showed much interest. The latter, instead, developed continuously during last century and entered into the compositional process directly, merging the theoretical level of music with the practical one.

The motivations for such a different fortune are not easy to be found. It is reasonable to suppose, however, that logical approaches are mainly related to musical *representation* while algebraic methods are connected with both musical representation and musical *creation*. For some reasons, it seemed appealing to apply symbolic logic to music even if this was not really effective.

Nonetheless, the situation is different for the last method described in chapter 2: the functional approach. This attempt evolved, in the last twenty years, in actual computer programs that helped musicians to create, understand and manipulate musical objects. Again, it seems that the process of music creation is fundamental to evaluate the *effectiveness* of any representation.

Creation is directly linked to one of the most important dimensions of music: *performance*. Symbolic representations of music are not really useful if they cannot deal with such a dimension.

The basic assumption of this reserch is, in fact, that musical representations must deal, in order to be effective, not only with music in its *static* stage as described by scores. They must be able to handle also the final stage of *performance*; in other words, symbols should be applied on objects that are *not* symbolic, such as signals.

For this reason, a new representation method has been proposed: the *theory of sound-types*. The core idea of the new method is the creation of *high level* concepts that group together entities that are at a lower level in the abstraction process. Since music happens in time, moreover, the new representation also expresses the temporal evolution of such high level concepts.

Musical information are incapsulated into two interacting entities: *types* and *rules*. While types represent concepts, rules encodes their evolution over

time. This approach takes inspiration from *theoretical* models (simple type theory and System F) but is realized by means of *computational* models (signal descriptions and statistical analysis).

This proposes the hypothesis that the inquiry into symbolic representations of music must change paradigm, moving from a theoretical approach to a computational one.

At the beginning of this work, some important question have been presented:

1. Which is the relationship between mathematical logic and musical logic?
2. Has the formalism based on *musical reasoning* something in common with logic formalism?
3. Can mathematical logic be useful to musicians, in order to clarify their reasoning?

Whether the theory of sound-types is able to answer to these questions is not easy to say. It is not wrong to say, however, that most of logical representations of music, in a sense, *failed*.

The paradigm change discussed above put new light on the relation between music and symbols, shifting the attention from abstract formalization to effective computation of musical parameters. This transformation is still evolving and many important results will be found only with future developments.

Next sections will focus on possible extensions of the proposed theory and will also outline future work in order to formulate a full framework for the analysis and the synthesis of music.

5.2 Generalization of the theory of sound-types

Sound-types seem to be promising entities to represent music because they are physically related to sound, are invertible and are also capable to represent formal relationships and hierarchies. It's important to point out, however, that the proposed algorithm is only a *possible* realization of a general idea (see (Cella 2009b)). The low-level features to be used can be many more

and so can be the clustering techniques. Other concatenation methods are also possible (see (Cont 2007), (Peeters 2007)) and, finally, the symbolic language for the sound representation is a matter of choice. The more expressive the language, the more the possible manipulations on the symbolic-level.

The whole problem of representing signals in symbolic ways is built of three major parts: a **back-end**, a **concatenation layer** and a **front-end**.

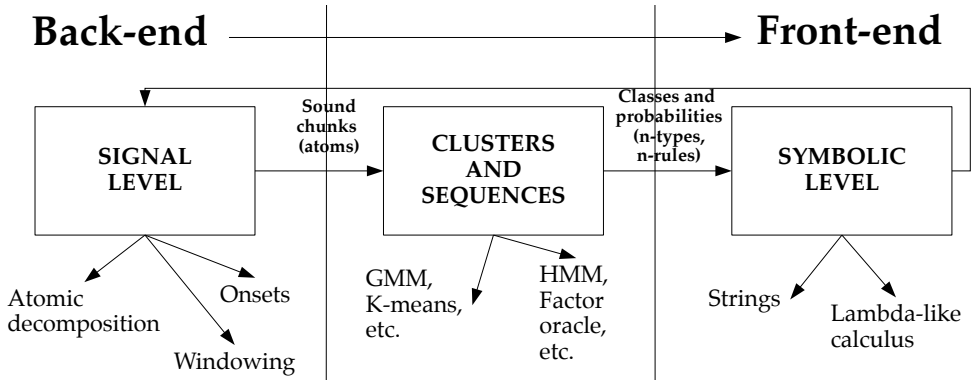


Figure 5.1: A global framework for the sound-type theory.

The main aim of the back-end is to provide chunks of sounds (atoms) to be subsequently analysed; this can be done simply by windowing or by using more complex techniques such as atomic decomposition or onsets separation. Once atoms have been defined, it's possible to look for *classes of equivalence* for sound and *concatenation* rules between classes in order to decompose a signal; this should be done in the second layer. Once classes and rules have been collected it's then possibile to represent them through a *grammar* of any symbolic language, either descriptive only or generative[1]; this is, finally, the aim of the front-end.

For each level there could be plenty of possibilities in terms of algorithms and techniques; it's therefore mandatory to define a sort of *interface* between levels, in order to have a modular system into which plug different tools on demand. Figure 5.1 depicts the described ideas.

[1]With the word *grammar*, here, we simply mean a corpus of syntactic rules that define any formal system.

In the context of this generalized framework, table 5.1 summarize the implemented features.

Table 5.1: Implemented techniques for each layer.

LAYER	**TECHINQUES**
Back-end	Windowing, onsets separation
Concatenation layer	Low-level features + GMM
	Low-level features + K-means
	Markov models
Front-end	Descriptive language (strings)

Obviously, one of the main lacking features in the current implementation is a powerful symbolic language. Representing signals with a sequence of labels is not enough to permit advanced manipulations in the symbolic-level.

An hypothetical language

A possible front-end language for the theory could take inspiration from the music calculus presented in section 2.4.2. With such a language, more transformations on musical signal would be possible; its grammar could be something similar to the following one:

$$
\begin{aligned}
type ::= & \; atom \\
& \mid [type_1 | type_2] \\
& \mid \left[\frac{type_1}{type_2}\right] \\
& \mid \lambda atom.type \; (\textbf{abstraction}) \\
& \mid (type_1 type_2) \; (\textbf{application}) \\
atom ::= & \; 0 \mid 1 \mid 2 \mid 3 \mid \ldots
\end{aligned}
$$

where $[type_1|type_2]$ represents the *successor* operator, $\lambda atom.type$ represents the type creation by classification and $\left[\frac{type_1}{type_2}\right]$ represents the summation of types. Many other operations can be defined in order to manipulate types, such as *time-shift*, *time-stretch* and so on.

If $s_1 = [01234]$ is the atomic decomposition of a signal and $\lambda x.x \rightarrow X$ is the **X-typization** operator (that create the type X give a type) and $\lambda x.x \rightarrow Y$ is the analogous **Y-typization** operator, then it's possible to express the following sentences:

$$\begin{aligned} s_1 &= [01234] \\ &= [\lambda x.x \rightarrow X0\ \lambda x.x \rightarrow Y1\ \lambda x.x \rightarrow Y2\ \lambda x.x \rightarrow Y3\ \lambda x.x \rightarrow X4] \\ &\Rightarrow_\beta [XYYYX] \end{aligned}$$

then, by defining the operator $\lambda x.x \rightarrow X1$ it's possible to translate them an higher level of abstraction:

$$\begin{aligned} s_1 &= [\lambda x.x \rightarrow X1X\ \lambda x.x \rightarrow X1Y\ \lambda x.x \rightarrow X1Y\ \lambda x.x \rightarrow X1Y\ \lambda x.x \rightarrow X1X] \\ &\Rightarrow_\beta [X1X1X1X1]. \end{aligned}$$

In this simple example, only types creation by means of classification has been considered and Markov models have not been represented. Moreover, a simply untyped λ-calculus have been used while many other possible languages could be adopted.

5.3 Future work

The **theory of sound-types** presented in chapter 4 is still in an early stage of development and needs expansions and improvements both in the symbolic-level and in the signal-processing level.

This is a partial list of possible relevant research directions; figure depicts some possible expansions of the theory.

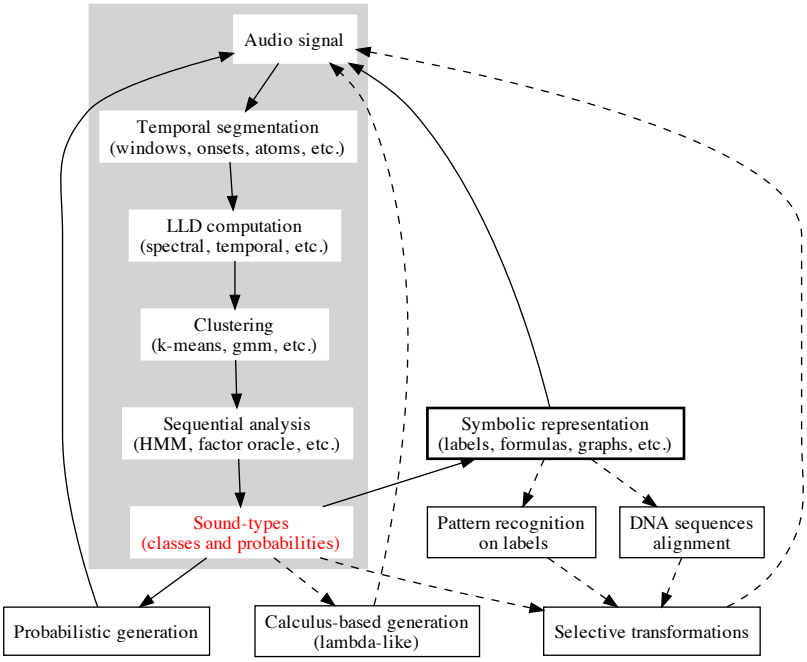


Figure 5.2: An overview of the possible expansions.

Typed languages

The representation on the symbolic-level is now a simple string while it should be done in a appropriate language that can handle types. For this reason an investigation in simply-typed languages should be done and a more expressive language for symbolic representation should be supported.

Higher levels

Higher levels are still not implemented; this could put into the game the transition probabilites and lead to a complete mathematical formulation of the theory.

Symbolic-level transformations

It should be possible to transform a sound working on the string of labels. For example, pitch-shift or time-stretch only selected types or extract some given types to perform semantic source separation.

Pattern recognition and alignment

It should be possible to analyse the symbolic-level representation created by the algorithm, in order to discover patterns by mean of dedicated techniques. Moreover, it could be interesting to *align* different representations of the same signal in order to discover structure information.

Applications and evaluation

More important applications of the proposed method should be found; this would also lead to the definition of more appropriated evaluation techniques in order to understand the real power of the proposed approach.

Appendix A

Source code

The following listings are the source code in C++ of the analysis and of the synthesis routines of the framework described in chapter 4. A typical configuration script to control the analysis process is also reported below.

The full implementation is made of other important parts, such as configuration, data exporting and so on, but they will not be listed here.

Listing A.1: The analysis routine

```
// analysis.h
//

#ifndef ANALYSIS_H
#define ANALYSIS_H

//#define USE_SLOW_FFT

#include "Matrix.h"
#include "algorithms.h"
#include "FFT.h"
#include "features.h"
#include "ModelingSpace.h"
#include <vector>
#include <cassert>
#include <cmath>
#include <stdexcept>
#include <iostream>
#include <cstring>

//#include "memCheck.h"

void analyse (float* data,  double* window, double* workspace,
    double* buffer,
        double* amp, double* oldAmp,
```

```
        double* freq, double* phi,
        ModelingSpace<double>& p, AbstractFFT<double>* ft,
        double sr, int nchnls, int maxBin,
        int totSamp, int& frames) {
  int pointer = 0;
  double tmp = 0, ctrd = 0, sprd = 0, f0 = 0;
  double winEnergy = 0;
  for (int i = 0; i < p.N; ++i) {
    winEnergy += (window[i] * window[i]);
  }

  do {
    memset (workspace, 0, sizeof (double) * 2 * p.N);
    memset (buffer, 0, sizeof (double) * p.N);

    //int r = p.N > totSamp - pointer ? totSamp - pointer : p.N;
    int r = pointer + p.N > (totSamp) ? pointer + p.N - (totSamp
        ) : 0;
    pointer -= r;
    for (int i = 0; i < p.N; ++i) {
      for (int j = 0; j < nchnls; ++j) {
        int pp = nchnls * (i + pointer) + j;
        // int pp = (nchnls * pointer) + (nchnls * i + j);

        workspace[2 * i] += (double) data[pp];
      }

      workspace[2 * i] *= window[i];
      workspace[2 * i] /= nchnls;
      buffer[i] = workspace[2 * i];
    }

#ifdef USE_SLOW_FFT
    fft<double> (workspace, p.N, -1);
#else
    ft->forward (workspace);
#endif
    convert<double> (workspace, amp, freq, phi, p.N, p.hop, sr);

    switch (p.scale) {
    case Features::LINEAR:
      // nothing to do for linear
```

```
  break;
case Features::LOG:
  for (int i = 0; i < p.N; ++i) {
    amp[i] = logAmplitude (amp[i]);
  }
  break;
case Features::POWER:
  for (int i = 0; i < p.N; ++i) {
    double a = amp[i];
    amp[i] = a * a;
  }
  break;
}

if (p.switches[Features::SPECTRAL_CENTROID]) {
  ctrd = speccentr<double> (amp, freq, maxBin);
  p.matrix[Features::SPECTRAL_CENTROID].push_back (ctrd);
}
if (p.switches[Features::SPECTRAL_SPREAD]) {
  sprd = specspread<double> (amp, freq, maxBin, ctrd);
  p.matrix[Features::SPECTRAL_SPREAD].push_back (sprd);
}
if (p.switches[Features::SPECTRAL_SKEWNESS]) {
  tmp = specskew<double> (amp, freq, maxBin, ctrd, sprd);
  p.matrix[Features::SPECTRAL_SKEWNESS].push_back (tmp);
}
if (p.switches[Features::SPECTRAL_KURTOSIS]) {
  tmp = speckurt<double> (amp, freq, maxBin, ctrd, sprd);
  p.matrix[Features::SPECTRAL_KURTOSIS].push_back (tmp);
}
if (p.switches[Features::SPECTRAL_FLUX]) {
  tmp = specflux<double> (amp, oldAmp, maxBin);
  p.matrix[Features::SPECTRAL_FLUX].push_back (tmp);
}
if (p.switches[Features::SPECTRAL_IRREGULARITY]) {
  tmp = specirr<double> (amp, maxBin);
  p.matrix[Features::SPECTRAL_IRREGULARITY].push_back (tmp);
}
if (p.switches[Features::SPECTRAL_DECREASE]) {
  tmp = specdecr<double> (amp, maxBin);
  p.matrix[Features::SPECTRAL_DECREASE].push_back (tmp);
}
```

```
    if (p.switches[Features::SPECTRAL_SLOPE]) {
      tmp = specslope<double> (amp, freq, maxBin);
      p.matrix[Features::SPECTRAL_SLOPE].push_back (tmp);
    }
    if (p.switches[Features::HIGH_FREQUENCY_CONTENT]) {
      tmp = hfc<double> (amp, maxBin);
      p.matrix[Features::HIGH_FREQUENCY_CONTENT].push_back (tmp)
          ;
    }
    if (p.switches[Features::TOTAL_ENERGY]) {
      tmp = energy<double> (buffer, p.N, winEnergy);
      p.matrix[Features::TOTAL_ENERGY].push_back (tmp);
    }
    if (p.switches[Features::ZERO_CROSSINGS]) {
      tmp = zcr<double> (buffer, p.N);
      p.matrix[Features::ZERO_CROSSINGS].push_back (tmp);
    }
    if (p.switches[Features::F0]) {
      // f0 = acfF0Estimate<double> (sr, buffer, result, p.N);
      f0 = fftF0Estimate<double> (amp, freq, maxBin);
      p.matrix[Features::F0].push_back (f0);
    }
    if (p.switches[Features::INHARMONICITY]) {
      double maxAmp = 0;
      tmp = inharmonicity<double> (amp, freq, maxBin, f0, sr,
          maxAmp);
      p.matrix[Features::INHARMONICITY].push_back (tmp);
    }
    pointer += p.hop;
    ++frames;
    if (r != 0) break;
  } while (true); //pointer <= (totSamp));
}

double clusterDispersion (ModelingSpace<double>& p, Matrix<
    double> data, int m,
              std::vector <double>& dispersions,
              DynamicMatrix<double>& classes) {
  double W = 0;
  for (int i = 0 ; i < p.clusters ; ++i) {
    double sumDist = 0;
    for (unsigned int j = 0 ; j < classes[i].size () ; ++j) {
```

```
      for (unsigned int k = j + 1; k < classes[i].size () ; ++k)
           {
        int p1 = classes[i][j];
        int p2 = classes[i][k];
        double d = p.distFun (data[p1], data[p2], m);

        sumDist += (d * d);
      }
    }

    double cd =  (sumDist / (2. * classes[i].size ()));
    dispersions.push_back (cd);
    W += cd;
  }
  // check
  if (std::isnan (W) || std::isinf (W)) return -1;
  else return W;
}

void getLabels (ModelingSpace<double>& p, Matrix<double>& data1,
     int* labels,
        Matrix<double>& centroids, std::vector<double>&
            dispersions,
        double& W, double& loglike, DynamicMatrix<double>&
            classes) {

  //FILE* stream = fopen ("work/data.txt","r");

  int n = data1.rows ();
  int m = data1.cols ();

  // PCA reduction
  int dimsAfterReduction = p.dimensions ? p.dimensions : m;
  if (p.dimensions) {
    Matrix<double> symmat (m, m);
    covmat (data1.data (), n, m, symmat.data ());
    double* evals = new double[m];
    double* interm = new double[m];

    tred2 (symmat.data (), m, evals - 1, interm - 1);
    tqli (evals - 1, interm - 1,  m, symmat.data ());
```

```
  // projection to eigenvector
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
      interm[j] = data1[i][j];
    }
    for (int k = 0; k < m; k++) {
      data1[i][k] = 0.;
      for (int k2 = 0; k2 < p.dimensions; k2++) {
        data1[i][k] += interm[k2] * symmat[k2][m - k]; // +
            1];
      }
    }
  }
  delete [] evals;
  delete [] interm;

  data1.resize (data1.rows (), dimsAfterReduction);
  p.pca.clear ();
  p.pca.resize (dimsAfterReduction);

  for (int i = 0; i < dimsAfterReduction; ++i) {
    std::vector<double> tt;
    for (int j = 0; j < n; ++j) {
      tt.push_back (data1[j][i]);
    }
    p.pca[i] = (tt);
  }
}

int h = (int) p.clusters;
if (p.clustAlgo == Features::KMEANS) {
  kmeans <double> (data1.data (), n, dimsAfterReduction, h,
           0.00001, labels, centroids.data ());
} else if (p.clustAlgo == Features::GMM) {
  // for (int h = 1 ; h < clusters ; ++h) {
  double ratio = (double) data1.rows () / h;
  // initial guess for gmm
  Matrix<double> means (h, dimsAfterReduction);
  for (unsigned int i = 0; i < means.rows (); ++i) {
    for (unsigned int j = 0; j < means.cols (); ++j) {
      int pos = i * ratio;
      // int pos = ((rand () % data1.rows ()) + (rand () % 10)
```

```
            ) % data1.rows ();
          means[i][j] = data1[pos][j];
        }
      }
      GMM<double> gmm (data1, means);
      for (int i = 0; i < (int) p.clusters; ++i) {
        for (int j = 0; j < dimsAfterReduction; ++j) {
          centroids[i][j] = gmm.means[i][j];
        }
      }

      for (int i = 0; i < n; ++i) {
        int max = 0;
        maximum (gmm.resp[i], p.clusters, max);
        labels[i] = max;
      }
      loglike = gmm.loglike;
    }
        //cout << h << " " << gap << endl;
      //}
    classes.clear ();
    classes.resize (h);
    for (int j = 0; j < n; ++j) {
      classes[labels[j]].push_back (j);
    }

    W = clusterDispersion  (p, data1, dimsAfterReduction,
        dispersions, classes);
}


void markovChain (Matrix<double>& markov, int* labels, int N) {
  for (int i = 0; i < N - 1; ++i) {
    ++markov[labels[i]][labels[i + 1]];
  }
}

#endif  // ANALYSIS_H

// EOF
```

Listing A.2: The synthesis routine

```
// synthesis.h
//

#ifndef SYNTHESIS_H
#define SYNTHESIS_H

#include "Matrix.h"
#include "BlockVocoder.h"
#include "ModelingSpace.h"
#include "FFT.h"
#include <vector>
#include <cstring>
#include <iomanip>

//#include "memCheck.h"

//#define GRANULAR_SYNTHESIS
//#define USE_SLOW_FFT

#define MAX_REPETITIONS (5)

struct SoundPacket  {
  int typeNumber;
  int instances;
};

double fran (double min, double max) {
  static double RMAX = 0x7fffffff;
  return ((max - min) * (rand () / RMAX) + min);
}

int wchoice (double* dist, int n) {
  double R = fran (0., 1.);
  for (int i = 0; i < n; ++i) {
    if (dist[i] >= R) return i;
  }

  return (int) fran (0, n);
}

void synthesize (float* data, double* window, double* workspace,
     double* workspace2,
```

```
        DynamicMatrix<double>& classes, Matrix<double>&
            centroids, int* labels,
        Matrix<double>& types,  double* synth,
        ModelingSpace<double>& p, AbstractFFT<double>* ft,
            Matrix<double>& probabilities,
        int nchnls, int totSamp, int frames) {

// types building
for (int t = 0; t < (int) p.clusters; ++t) {
  // compute distances from the center of the cluster
  std::vector <double> distances (classes[t].size ());
  double totDistance = 0;

  for (unsigned int j = 0 ; j < classes[t].size (); ++j) {
    std::vector <double> features;
    if (p.dimensions == 0) { // pca not applied
      for (unsigned int i = 0 ; i < p.descriptors.size (); ++i
          ) {
        if (p.switches[i] == true) {
          features.push_back (p.matrix[i][classes[t][j]]);
        }
      }
    } else { // pca applied
      for (unsigned int i = 0; i < p.pca.size (); ++i) {
        features.push_back (p.pca[i][classes[t][j]]);
      }
    }

    double d = p.distFun (&features[0], centroids[t], features
        .size ());
    distances[j] = d;
    totDistance += d;
  }

  if (p.algorithm == Features::RANDOM) {
    if (classes[t].size ()) { // skip empty centriods
      int rv = (int) rand ();
      double qr = (double) rv / RAND_MAX;
      qr *= classes[t].size ();
      int q = (int) qr;
      int pointer = (int) ((double) p.hop *  classes[t][q]);
      int r = pointer + p.N > (totSamp) ? pointer + p.N - (
```

```
        totSamp) : 0;
      pointer -= r;
      for (int i = 0; i < p.N; ++i) {
        double sample = 0;
        for (int j = 0; j < nchnls; ++j) {
          int pos = (nchnls * pointer) + (nchnls * i + j);
          sample += (double) data[pos];
        }
        types[t][i] = (sample / nchnls) * window[i];
      }
    }
  } else if (p.algorithm == Features::AVERAGE || p.algorithm
      == Features::WEIGHTED) {
    for (unsigned int q = 0; q < classes[t].size (); ++q) {
      double cnorm = 1. / classes[t].size ();
      if (p.algorithm == Features::WEIGHTED) cnorm = distances
          [q] / (totDistance != 0 ? totDistance : 1);
      int frame = classes[t][q];
      int pointer = (int) ((double) p.hop * frame);
      //int r = p.N > totSamp - pointer ? totSamp - pointer :
          p.N;
      int r = pointer + p.N > (totSamp) ? pointer + p.N - (
          totSamp) : 0;
      pointer -= r;
      for (int i = 0; i < p.N; ++i) {
        double sample = 0;
        for (int j = 0; j < nchnls; ++j) {
          sample += (double) data[(nchnls * pointer) + (nchnls
               * i + j)];
        }
        types[t][i] += (sample / nchnls * cnorm) * window[i];
      }
    }
  } else if (p.algorithm == Features::CLOSEST) {
    if (classes[t].size ()) { // skip empty centriods
      int q = 0;
      minimum (&distances[0], distances.size (), q);

      int pointer = (int) ((double) p.hop *  classes[t][q]);
      //int r = p.N > totSamp - pointer ? totSamp - pointer :
          p.N;
      int r = pointer + p.N > (totSamp) ? pointer + p.N - (
```

```
            totSamp) : 0;
          pointer -= r;
          for (int i = 0; i < p.N; ++i) {
            double sample = 0;
            for (int j = 0; j < nchnls; ++j) {
              sample += (double) data[(nchnls * pointer) + (nchnls
                   * i + j)];
            }
            types[t][i] = (sample / nchnls) * window[i];
          }
        }
      }
    }

    // probabilistic generation
    Matrix<double> dist (probabilities.rows (), probabilities.rows
         ());
    if (p.decompose == 2) {
      unsigned int k = 0;
      for (unsigned int i = 0; i < probabilities.rows (); ++i) {
        for (unsigned int j = 0; j < probabilities.rows (); ++j) {
          for (dist[i][j] = k = 0; k <= j; k++) {
            dist[i][j] += probabilities[i][k];
          }
        }
      }
    }

    int outhop = (int) ((double) p.hop * p.stretch);

  #ifdef GRANULAR_SYNTHESIS
    // FIMXE: not updated for probabilistic generation
    if (!p.interp) {
      // granular synth
      for (int t = 0; t < frames ; ++t) {
        int jitter = p.jitter == 0 ? p.jitter : rand () % p.jitter
            ;
        int pointer = (t * outhop) + jitter;

        for (int j = pointer; j < pointer + p.N; ++j) {
          //synth[j] += data[j] * window[j - pointer] / 4;
          synth[j] += types[labels[t]][j - pointer]/* * window[j -
```

```
            pointer]*/ * p.normalization;
        }
      }
    } else {
      int pointer = 0;
      for (unsigned int t = 0; t < packets.size () - 1; ++t) {
        SoundPacket pk = packets[t];
        SoundPacket pk_next = packets[t + 1];
        double interp = 1. / pk.instances;
        for (int i = 0; i < pk.instances; ++i)  {
          int jitter = p.jitter == 0 ? p.jitter : rand () % p.
              jitter;
          pointer += outhop;
          double alpha = (double) i * interp;
          for (int j = pointer; j < pointer + p.N; ++j) {
            double sample = (1. - alpha) * types[pk.typeNumber][j
                - pointer] +
              alpha * types[pk_next.typeNumber][j - pointer];
            synth[j + jitter] +=  sample * /*window[j - pointer] *
                */ p.normalization;
          }
        }
      }
    }
  #else
    BlockVocoder<double> pv (p.N);
    int NN = p.N << 1;

    double norm = 0;
    for (int i = 0; i < p.N; ++i) {
      norm += (window[i]);
    }
    norm = 1. / ((norm * ((p.N /2) / outhop)) * (p.pitch < 1 ? p.
        pitch : 1));

    if (!p.interp) {
      std::cout << "not interp " << (p.decompose == 1 ? "rebuild"
          : "generate");

      int state = 0;
      int prevState = 0;
      int delooper = 0;
```

```
    // phase vocoder
    for (int t = 0; t < frames; ++t) {
      int jitter = p.jitter == 0 ? p.jitter : rand () % p.jitter
          ;
      int pointer = (t * outhop) + jitter;

      state = (p.decompose == 2 ? wchoice (dist[state],
          probabilities.rows ()) : labels[t]);
      if (state == prevState && p.decompose == 2) delooper++;

      if (delooper == MAX_REPETITIONS) {
        state = (int) fran (0, probabilities.rows ());
        delooper = 0;
      }
      prevState = state;


      memset (workspace, 0, sizeof (double) * NN);
      for (int j = 0; j < (int) p.N; ++j) {
        workspace[2 * j] = types[state][j];
        workspace[2 * j + 1] = 0;
      }

      // transformation: in -> workspace, out -> workspace2
      pv.process (workspace, workspace2, p.pitch, p.hop, outhop,
           p.C, p.threshold);

      // overlapp-add
      for (int j = 0; j < p.N; ++j) {
        synth[j + pointer] += (workspace2[2 * j] * window[j] *
            norm) * p.normalization;
      }
    }
  } else {
    std::cout << "interp " << (p.decompose == 1 ? "rebuild" : "
        generate");

    std::vector<SoundPacket> packets;
    int lastType = -1;
    int state = 0;
    int prevState = 0;
```

```
int delooper = 0;

for (int i = 0; i < frames; ++i) {
  if (packets.size () == 0 || labels[i] != lastType) {
    state = (p.decompose == 2 ? wchoice (dist[state],
        probabilities.rows ()) : labels[i]);
    if (state == prevState && p.decompose == 2) ++delooper;

    if (delooper == MAX_REPETITIONS) {
      state = (int) fran (0, probabilities.rows ());
      delooper = 0;
    }
    prevState = state;

    SoundPacket pk;
    pk.typeNumber = state;
    pk.instances = 1;
    packets.push_back (pk);
  } else {
    ++packets[packets.size () - 1].instances;
  }
  lastType = state;
}
// last packet guard point for interpolation
packets.push_back (packets[packets.size () - 1]);

int pointer = 0;
for (unsigned int t = 0; t < packets.size () - 1; ++t) {
  SoundPacket pk = packets[t];
  SoundPacket pk_next = packets[t + 1];
  double interp = 1. / pk.instances;
  for (int i = 0; i < pk.instances; ++i)  {
    int jitter = p.jitter == 0 ? p.jitter : rand () % p.
        jitter;
    pointer += outhop;

    double alpha = (double) i * interp;
    memset (workspace, 0, sizeof (double) * NN);
    for (int j = 0; j < (int) p.N; ++j) {
      workspace[2 * j] = (1. - alpha) * types[pk.typeNumber
          ][j] +
        alpha * types[pk_next.typeNumber][j];
```

```
          workspace[2 * j + 1] = 0;
        }

        // transformation: in -> workspace, out -> workspace2
        pv.process (workspace, workspace2, p.pitch, p.hop,
            outhop, p.C, p.threshold);

        // overlapp-add
        for (int j = 0; j < p.N; ++j) {
          synth[j + pointer + jitter] += (workspace2[2 * j] *
              window[j] * norm) * p.normalization;
        }

      }
    }
  }
#endif

}

#endif  // SYNTHESIS_H

// EOF
```

Listing A.3: Configuration script

```
;
; clusters - configuration file
;

analysis.wintype      hanning
analysis.winsize      4096
analysis.hopsize      512
analysis.maxfreq      5512.5
analysis.ampscale     log
analysis.descriptors    centroid spread skewness
analysis.modeling     1
analysis.savedesc     1

clustering.algorithm    gmm
clustering.centroids    .1
clustering.dimensions   0
```

```
clustering.showlabels   0

decomposition.performance rebuild
decomposition.distance    taxicab
decomposition.algorithm   average
decomposition.stretch   1
decomposition.pitch     1
decomposition.envelope    0
decomposition.threshold   0
decomposition.interpolate 1
decomposition.normalization .75
decomposition.jitter    0

global.verbose        0

; * help *
;
; - max analysis frequency is nyquist
; - available features: centroid spread skewness kurtosis
;   irregularity slope decrease energy hfc zcr flux f0
    inharmonicity
; - available windows: hanning hamming blackman bartlett
; - available amplitude scales: lin log pow
; - available clustering algorithms: gmm, kmeans
; - the number of centroids is a ratio of the number of frames;
;   max is 1, 0 means no clustering
; - if dimensions are 0 pca reduction is not applied, otherwise
    is the # of pcas
; - types of performances are: none, rebuild, generate
; - available distance measures: euclid mahalanobis taxicab
    cosim
; - available algorithms for decomposition: average weighted
    random closest
; - pitch and stretch are ratios of original file
; - threshold is used for denoising
; - envelope is the number of coefficient for envelope
    preservation
; - jitter is specified in samples
```

Appendix B

Related publications

The following list collects all the publications by the author that are related to this research: some of the results presented here have already been published.

- Carmine E. Cella, *Sound-types: a new framework for symbolic sound analysis and syntehsis*, ICMC 2011, Huddersfield, United Kingdom;
- Carmine E. Cella, *Harmonic Components Extraction in Recorded Piano Tone*, 128th AES conference London, United Kingdom, 2010;
- Carmine E. Cella, *Nuovi approcci alla struttura armonica: caleidocicli e mosaici tricordali (book chapter)*, in Luigi Verdi - *Caleidocicli musicali*, 2nd edition, 2010 Milano, Rugginenti;
- Carmine E. Cella, *Towards a Symbolic Approach to Sound Analysis*, Second international conference on Mathematics and computation for music Yale University, New Haven, Springer, 2009;
- J.J. Burred, C. E. Cella, G. Peeters, A. Röbel and D. Schwarz. *Using the SDIF Sound Description Interchange Format for Audio Features*, Proc. International Conference on Music Information Retrieval (ISMIR), Philadelphia, USA, September 2008;
- Carmine E. Cella, *Review of Caleidocicli musicali by Luigi Verdi*, Musica theorica Spectrum, Fall 2006, Ed. Curci Milano (Italy);
- Carmine E. Cella, *Il compositore cieco*, Rivista umbra di musicologia, n. 50 2006/1, pagg. 17 27 Perugia (Italy);
- Carmine E. Cella, *Il semplice sistema*, Rivista umbra di musicologia, n. 49 2005/2, pagg. 43 51 Perugia (Italy);

- Carmine E. Cella, *Sulla struttura logica della musica*, Rivista umbra di musicologia, n. 48 2005/1, pagg. 3 57, Perugia (Italy);
- Carmine E. Cella, F. Paolinelli, *Sintesi per stati e visualizzazione del processo compositivo*, La Terra Fertile, Proceedings, 2000, pagg. 87-89, L'Aquila (Italy).

List of Figures

List of Tables

Bibliography

Adrien, J. M., Arfib, D., D'Autilia, R. and et al.: 1991, *Representations of musical signals, edited by G. De Poli, and A. Piccialli, and C. Roads*, The MIT Press.

Amatriain, X., Bonada, J., Loscos, A. and Serra, X.: 2002, *Spectral processing, in DAFX - Digital Audio Effects, edited by U. Zölzer*, John Wiley & Sons.

Andreatta, M.: 2003, Méthodes algébriques dans la musique et la musicologie du XXéme siécl: aspects théoriques, analytiques et compositionnels, *PhD thesis, EHESS/IRCAM*, Paris - FRANCE.

Aqvist, L.: 1979, Music from a set-theoretical point of view, *Interface, 2*.

Assayag, G.: 2002, *Mathematics and music: a Diderot mathematical forum. Edited by G. Assayag, and H. G. Feichtinger, and J. F. Rodrigues*, Springer.

Babbit, M.: 1950, Schönberg et son école. Qu'est ce que la musique de douze son?, edited by rené leibowiz, *Journal of american musicological society, 3/1*.

Babbit, M.: 1955, Some aspect of twelve-tone composition, *The score, 12*.

Babbit, M.: 1960, Twelve-tone invariants as a compositional determinant, *The musical quarterly, 46/2*.

Babbit, M.: 1961, Set structure as a compositional determinant, *Journal of music theory, 5/2*.

Barendregt, H.: April 1991, Introduction to generalized type systems, *Journal of functional programming 1(2)*.

Bel, B.: 1989, Pattern grammars in formal representations of musical structures, *11th International joint conference on Artificial Intelligence*.

Bell, A. J. and Sejnowski, T.: 1996, Learning the higher order structure of a natural sound, *Network: Computation in neural systems, 7*.

Bello, J. P., Daudet, L., Abdallah, S., Duxbury, C., Davies, M. and Sandler, M. B.: 2005, A tutorial on onset detection in music signals, *Proc. IEEE Transactions on speech and audio processing, 13(5), part 2:1035-1047*.

Bishop, C. M.: 2006, *Pattern recognition and machine learning*, Springer.

Blevis, E., Jenkins, M. and Robinson, E.: 1989, On seeger's music logic, *Interface, 18*.

Bobbit, R.: 1959, The physical basis of intervallic quality, *Journal of music theory, 3*.

Bregman, A.: 1990, *Auditory scene analysis*, MIT Press.

Brown, G. J.: 1992, Computational auditory scene analysis: a representational approach, *PhD thesis, University of Sheffield*, UK.

Brown, G. J.: January 1991, Calculation of a constant Q spectral tranform, *Journal of the acoustical society of America, 89(1)*.

Browne, R.: 1974, Review of the structure of atonal music by Allen Forte, *Journal of music theory, 18/2*.

Burges, C. J. C.: 1998, *A tutorial on support vector machines for pattern recognition, in data mining and knowledge discovery, edited by G. I. Webb*, Kluwer Academic Publishers.

Burred, J. J., Röbel, A. and Rodet, X.: 2006, An accurate timbre model for musical instruments and its application to classification, *Workshop on learning the semantics of audio signals (LSAS)*, Athens - GREECE.

Casey, M.: 2002, *Sound classification and similarity tools, in Introduction to MPEG-7, edited by P. Salembier, and B.S. Manjunath, and T. Sikora*, John Wiley.

Casey, M. A.: 1998, Auditory group theory with applications to statistical basis methods for structured audio, *Phd thesis, Massachusetts institute of technology*.

Casey, M. A.: October 2005, Acoustic lexemes for organizing internet audio, *Contemporary music review*.

Cella, C. E.: 2004, Sulla struttura logica della musica, *Rivista umbra di musicologia, 48*.

Cella, C. E.: 2009a, A state-of-the-art report on the phd, *Internal report, IRCAM, Paris*.

Cella, C. E.: 2009b, Towards a symbolic approach to sound analysis, *Mathematics and Computation for Music Conference (MCM)*, Yale University - New Haven.

Chrisman, J.: n.d., Identification and correlation of pitch-sets, *Journal of music theory, 15*.

Church, A.: 1985, *The calculi of lambda-conversion*, Princeton University Press.

Clough, J.: 1965, Pitch set equivalence and inclusion, *Journal of music theory, 9/1*.

Clough, J.: 1979, Aspect of diatonic set, *Journal of music theory, 23/1*.

Comon, P.: 1994, Indipendent component analysis - a new concepts?, *Signal processing, 36:287-314*.

Cont, A.: 2007, Audio Oracle: a new algorithm for fast learning of audio structures, *International Computer Music Conference (ICMC)*.

Cont, A., Dubnov, S. and Assayag, G.: December 2008, On the information geometry of audio streams with applications to automatic structure analysis, *Proc. IEEE audio, speech, and language processing*.

D. Arfib, F. K. and Zölzer, U.: 2002, *Source-filter processing, in DAFX - Digital Audio Effects, edited by U. Zölzer*, John Wiley & Sons.

Daubechies, I.: 1988, Time-frequency localization operators: a geometric phase space approach, *IEE Transactions on information theory, 34:605-612*.

de Cheveigné, A. and Kawahara, H.: April 2002, YIN, a fundamental frequency estimator for speech and music, *Journal of the acoustical society of America, 111(4):1997-1930*.

Duda, R. O., Hart, P. E. and Stork, D. G.: 2000, *Pattern classification*, Wiley-Interscience, second edition.

Duda, R. O., Lyon, R. F. and Stanley, M.: 1990, Correlograms and the separation of sounds, *Proc. Asilomar conference on signals, systems and computers*, Pacific Grove - USA.

Ellis, D. and Rosenthal, D.: 1995, Mid-level representations for computational auditory scene analysis, *International joint conference on artificial intelligence*.

Eschmann, K. H.: 1968, *Changing form in modern music, 2nd edition*, E. C. Schirmer Music Co.

Fletcher, N. H. and Rossing, T. D.: 1998, *The physics of musical instruments*, Springer.

Foote, J.: March 1997, A similarity measure for automatic audio classification, *Proc. AAAI 1997 Spring symposium on intelligent integration and use of text, image, video, and audio corpora.*, Stanford - USA.

Forte, A.: 1964, A theory of set-complexes for music, *Journal of music theory, 8*.

Forte, A.: 1965, A theory of set-complexes for music, *Journal of music theory, 9/1*.

Forte, A.: 1977, *The structure of atonal music*, Yale University Press.

Forte, A.: 1985, Pitch-class set genera and the origin of modern harmonic species, *Journal of music theory, 29*.

Galas, T. and Rodet, X.: 1991, Generalized discrete cepstral analysis for deconvolution of source-filter systems with discrete spectra, *Proc. IEEE workshop on applications of signal processing to audio and acoustic (WASPAA)*, New Paltz - USA.

Gerhard, R.: 1952, Tonality in twelve-tone music, *The score, 6*.

Gerhard, R.: 1954, Reply to George Perle, *The score, 9*.

Gingerich, L.: n.d., Another method for teaching exachordal combinatoriality, *In theory only, 7/3*.

Girard, G.-Y.: 2006, *Le Point Aveugle: Tome 1. Cours de Logique, Vers la perfection.*, Hermann.

Goodwin, M.: 1998, *Adaptive signal models*, Kluwer Academic Publishers.

Goodwin, M. and Vetterli, M.: 1997, Atomic decompositions of audio signal, *Proc. IEEE Audio Signal Processing Workshop*.

Górecki, L. and Domański, M.: 2005, Adaptive dictionaries for matching pursuit with separable decomposition, *European Signal Processing Conference*.

Grey, J. M.: 1977, Multidimensional perceptual scalig of musical timbre, *Journal of the acoustical society of America, 61:1270-1277*.

Hamerly, G. and Elkan, C.: December 2003, Learning the k in k-means, *Seventeenth annual conference on neural information processing systems (NIPS)*.

Hanson, H.: 1960, *Harmonic materials of modern music: resources of the tempered scale*, Irvington Pub.

Headlam, D.: 1983, Teaching hexachordal combinatoriality, *In theory only, 7/3*.

Headlam, D.: 1984, Recognizing pitch-class collections of limited transposition, *In theory only, 7/4*.

Herrera, P., Peeters, G. and Dubnov, S.: 2003, Automatic classification of musical instrument sounds, *Journal of new music research, 32(1)*.

Hess, W.: 1983, *Pitch determination of speech signals*, Springer.

Howe, H.: 1965, Some combinational properties of pitch structures, *Perspective of new music, 4/1*.

J. F. Cardoso, J. D. and Patanchon, G.: 2003, Indipendent component analysis of the cosmic microwave background, *Proc. International symposium on independent component analysis and blid signal separation (ICA)*, Nara - JAPAN.

Jedrzejewski, F.: 2006, *Mathematical theory of music*, Editions Delatour & Ircam-Centre Pompidou.

Jensen, A. and la Cour-Harbo, A.: 2001, *Ripples in mathematics*, Springer.

Jensen, K.: 2002, The timbre model, *Proc. 144th meeting of the acoustical society of America*, Cancùn - Mexico.

Jolliffe, I. T.: 2002, *Principal component analysis*, Springer.

Kaper, H. G. and Tipei, S.: 1999a, An abstract approach to music, *Argonne preprint ANL/MCS-P748-0399*.

Kaper, H. G. and Tipei, S.: 1999b, Formalizing the concept of sound, *Proc. 1999 Int'l Computer Music Conference*, Beijing, CHINA.

Keith, M.: 1991, *From polychords to pólya: adventures in musical combinatorics*, Vinculum Press.

Klapuri, A.: 2006, *Signal processing methods for the transcription*, Springer.

Kraft, L.: 1971, The music of Geroge Perle, *The musical quarterly, 57/3*.

Kunst, J.: 1976, Making sense in music I. The use of mathematical logic, *Interface, 5*.

Langer, S. K.: 1929, A set of postulates for the logical structure of music, *The monist, 39 (4)*.

Laroche, J. and Dolson, M.: 1999, New phase-vocoder techniques for real-time pitch shifting, chorusing, harmonizing, and other exotic audio modifications, *Journal of the audio engineering society, 47*.

Laske, O., Kugel, P., Smoliar, S. W. and et al.: 1992, *Understanding music with AI: perspectives on music cognition, edited by M. Balaban, and K. Ebcioglu, and O. E. Laske*, AAAI Press.

Leman, M.: 2002, Expressing coherence of musical perception in formal logic, *Mathematics and music: a Diderot mathematical forum*, pp. 184–198.

Lewicki, M. S. and Sejnowski, H. S.: n.d., Learning overcomplete representations, *Neural computation, 12(2):337-365*.

Lewin, D.: 1959, Intervallic relations between two collections of notes, *Journal of music theory, 3*.

Lewin, D.: 1960, The intervallic content of a collection of notes, *Journal of music theory, 4*.

Lewin, D.: 1962, A theory of segmental association in twelve-tone music, *Perspectives of new music, 1*.

Lewin, D.: 1977, Forte's interval vector, my interval function and Regener's common-note function, *Journal of music theory, 21*.

Lewin, D.: 2007, *Generalized musical intervals and transformations*, Oxford University Press.

Lord, C. H.: 1981, Intervallic similarity relation in atonal set analysis, *Journal of music theory, 25*.

Mallat, S. and Zhang, Z.: 1993, Matching pursuits with time-frequency dictionaries, *Proc. IEEE Transactions on signal processing, 41(3397:3415)*.

Marsden, A.: 2007, Timing in music and modal temporal logic, *Journal of mathematics and music*, Vol. 1, No. 3.

Martino, D.: 1961, The source set and its aggregate formations, *Journal of music theory, 5/2*.

McAulay, R. J. and Quatieri, T. F.: 1984, Magnitude-only reconstruction using a sinusoidal speech model, *Proc. IEEE international conference on acoustics, speech, and signal processing (ICASSP)*, New York - USA.

Mead, A.: 1984, A practical method for dealing with unordered pitch-class set collection, *In theory only, 7/4-5*.

Moore, F. R.: 1990, *Elements of computer music*, Prentice hall.

Morris, R. and Alegant, B.: 1988, The even partition in twelve-tone music, *Music theory spectrum, 10*.

Morris, R. D.: 1982, Set groups, complementation, and mapping among pitch-class sets, *Journal of music theory, 26*.

Morris, R. D.: 1987, *Composition with pitch-classes: a theory of compositional design*, Yale University Press.

Morris, R. D. and Street, D.: 1977, A general theory of combinatoriality and the aggregate, *Perspectives of new music, 16/1*.

Morton, I. A.: 1960, Numerical orders in triadic harmony, *Journal of music theory, 4*.

Muhr, M. and Granitzer, M.: 2009, Automatic cluster number selection using a split and merge K-means approach, *In Proc. of DEXA Workshops*, Linz - AUSTRIA.

O'Grady, P. D. and Pearlmutter, B. A.: 2004a, Hard-LOST: modified k-means for oriented lines, *Proc. Irish signals and systems conference*, Belfast - UK.

O'Grady, P. D. and Pearlmutter, B. A.: 2004b, Soft-LOST: EM on a mixture of oriented lines, *Proc. International conference on independent component analisys and blind signal separation (ICA)*, Granada - Spain.

Oppenheim, A. V. and Schafer, R. W.: 2009, *Discrete-time signal processing*, Prentice hall, 3rd edition.

Oppenheim, A. V., Schafer, R. W. and Buck, J. R.: 1999, *Discrete-time signal processing*, Prentice-hall, 2nd edition.

Orlarey, Y., Fober, D., Letz, S. and Bilton, M.: 1994, Lambda calculus and music calculi, *Proc. International computer music conference*.

Peeters, G.: 2007, Sequence representation of music structure using higher-order similarity matrix and maximum-likelihood approach, *ISMIR Wien*.

Peeters, G.: April 2004, A large set of audio features for sound description (similarity and classification) in the CUIDADO project, *CUIDADO I.S.T. Project Report*.

Peeters, G.: October 2003, Automatic classification of large musical instrument databases using hierarchical classifiers with inertia ratio maximization, *AES 115th convention*, New York - USA.

Peeters, G. and Rodet, X.: 2002, Automatically selecting signal descriptors for sound classification, *ICMC*, Goteborg - SWEDEN.

Pelleg, D. and Moore, A.: 2000, X-means: extending K-means with efficient estimation of the number of clusters, *Proc. of the 17th International Conference on Machine Learning*, Morgan Kaufmann, San Francisco - USA.

Perle, G.: 1954, The possible chords in twelve-tone music, *The score, 9*.

Perle, G.: 1957, The hexachord and its relation to the 12-tone row, by George Rochberg, *Journal of american musicological society, 10/1*.

Perle, G.: 1964, An approach to simultaneily in twelve-tone music, *Perspectives of new music, 8/2*.

Perle, G.: 1991, *Serial composition and atonality: an introduction to the music of Schoenberg, Berg, and Webern, 6th edition, revised*, Berkeley: University of California Press.

Perle, G.: 1996, *Twelve-Tone Tonality, 2nd edition*, Berkeley: University of California Press.

Press, W. H., Teukolsky, S. A., Vetterling, W. T. and Flannery, B. P.: 2007, *Numerical recipes: the art of scientific computing*, Cambridge University Press, 3rd edition.

Rabiner, L. R.: February 1989, A tutorial on hidden markov models and selected applications in speech recognition, *Proc. IEEE, 77(2)*.

Rahn, J.: 1980, *Basic atonal theory*, MacMillan Publishing Company.

Regener, A.: 1974, On Forte's theory of chords, *Perspective of new music, 13/1*.

Röbel, A. and Rodet, X.: 2005, Efficient spectral envelope estimation and its application to pitch shifting and envelope preservation, *Proc. International conference on digital audio effetcts (DAFX)*, Madrid - SPAIN.

Rochberg, G.: 1955, *The hexachord and its relation to the 12-tone row*, Theodore Presser Co.

Rochberg, G.: 1959, The harmonic tendency of the hexachord, *Journal of music theory, 3*.

Rojas, R.: 1997, A tutorial introduction to the lambda calculus, FU Berlin, WS-97/98.

Rothgeb, J.: 1967, Some ordering relationships in twelve-tone system, *Journal of music theory, 11/2*.

Rouse, S.: 1984, Exachords and their trichordal generator. An introduction, *In theory only, 8*.

San, H. V.: 1973, Sundry notes introductory to the theoretical mechanics of mathematical music, *Interface, 2*.

Schäfer, R. M.: 1985, *Il paesaggio sonoro*, BMG Ricordi Publications.

Schat, P.: 1993, *The tone clock*, Routledge.

Schwarz, D.: 1998, Spectral envelopes in sound analisys and synthesis, *Master's thesis*, Universitat Stuttgärt / IRCAM, Paris.

Serra, X.: 1997, *Musical sound modeling with sinusoids plus noise, in musical signal processing, edited by C. Roads, A. Picialli, and G. De Poli*, Swets & Zeitlinger.

Smith, J. O.: 2005, Physical audio signal processing: for virtual musical instruments and digital audio effects, *Center for computer research in music and acoustics (CCRMA)*, Stanford University.

Smith, J. O.: 2007, Spectral audio signal processing, march 2007 version, *Center for computer research in music and acoustics (CCRMA)*, Stanford University.

Smith, L. I.: February 2002, A tutorial on principal components analysis.

Solomon, L.: 1982, The list of chords, their properties and use in analysis, *Interface, 11/2*.

Starr, D.: 1978, Sets, invariance and partitions, *Journal of music theory, 22*.

Starr, D. and Morris, R.: 1974, The structure of all-interval series, *Journal of music theory, 18/2*.

Sutton, R.: 1987, Howard Hanson. A set theory pioneer, *Sonus, 8/1*.

Teitelbaum, R.: 1965, Intervallic relations in atonal music, *Journal of music theory, 9*.

Tibshirani, R., Walther, G. and Hastie, T.: 2000, Estimating the number of clusters in a data set via the gap static, *Technical report*, Stanford University - USA.

Travis, R.: 1959, Towards a new concept of tonality, *Journal of music theory, 3*.

Vauclain, C.: 1965, An experiment in musical texture, *The musical quarterly, 51/2*.

Vercoe, B. L., Gardner, W. G. and Scheirer, E. D.: 1998, Structured audio: creation, transmission and rendering of parametric sound representations, *Proceedings of the IEEE, 86, no. 5*.

Verdi, L.: 1998, *Organizzazione delle altezze nello spazio temperato*, Diastema editrice.

Verrall, J.: 1962, A method for finding symmetrical exachords in serial form, *Journal of music theory, 6/2*.

Vinet, H.: 2003, The representation levels of music information, *Lecture notes in computer science, Springer Verlag*, Vol. 2771.

Wessel, D.: 1979, Timbre space as musical control structure, *Computer music journal, 3(2):45-52*.

Wickerhauser, M. V.: 1996, *Adapted wavelet analysis from theory to software*, A K Peters/CRC Press.

Wilcox, H. J.: 1983, Group tables and the generalized hexachord theorem, *Perspectives of new music, 21*.

Wilding-White, R.: 1961, Tonality and scale theory, *Journal of music theory, 5/2*.

Zölzer, U., Dutilleux, P., Rocchesso, D. and et al.: 2005, *DAFX - Digital Audio Effects, edited by Zölzer*, John Wiley & Sons, LTD.

Index