



FH | JOANNEUM  
University of Applied Sciences



## Master's thesis

---

# Generative Sequencing

*Experimenting with Algorithmic Improvisation in Sound Design*

**Mag. phil. Reinhold Aschbacher**

---

**Supervisor: Mag. art. Dr. phil. Gerhard Nierhaus**

This thesis is submitted for the degree: Master of Arts (M.A.)

Field of Study: Sound Design (V 066 778)

Responsible institutions:

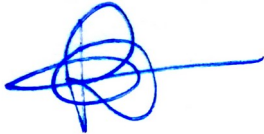
FH JOANNEUM University of Applied Sciences  
Institute of Design & Communication

University of Music and Performing Arts Graz  
Institute of Electronic Music and Acoustics

Graz, January 2021

## Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material that has been quoted either literally or by content from the used sources.

A handwritten signature in blue ink, consisting of several overlapping loops and a long horizontal stroke extending to the right.

Reinhold Aschbacher, January 2021

## **Acknowledgments**

I would like to thank my supervisor, Dr. Gerhard Nierhaus, whose expertise was invaluable throughout the writing process of this thesis. I would also like to thank all those who were teaching in the sound design master's course in cooperation between the FH-Joanneum and the University of Music and Performing Arts Graz. Especially Dr. Franz Zotter and DI Marian Weger from the Institute of Electronic Music and Acoustics have often been a great help in clarifying questions concerning the mathematical background, digital signal processing, and programming.

## **Abstract:**

The present thesis investigates methods of designing musical pieces, sound performances, and installations utilizing temporal sequencing and the generation of structure by different algorithmic design approaches. The creative goal described in this work is the development of a software setup suitable for generative sequencing and the description of some practical examples using this setup.

The introduction asks for the genesis of structure and meaning in human perception and consciousness through grouping principles, to derive some basic factors of audible structure perception and experience. It also gives a brief thematic and historical overview of musical sequencing and the use of some generative methods in art and design. A short introduction to the software Puredata is added, as it will be used in the further course of this work.

The main part of the present work focuses on the design of a generative sequencing system, which will be discussed in its essential components. The generative setup will therefore be outlined in its structure and functionality. On this basis, the practical implementation of its different components in Puredata will be described. Some application examples will be shown, which illustrate the described system for generative sequencing. Thereby I will focus on the aspects relevant for understanding how different generative models and methods can be used in the practice of sound design.

## **Kurzfassung:**

Die vorliegende Arbeit untersucht Methoden der Gestaltung von Klangstücken, Performances und Klanginstallationen mittels zeitlicher Sequenzierung und der Generierung von Struktur durch verschiedene algorithmische Gestaltungsansätze. Das in dieser Arbeit beschriebene gestalterische Ziel ist die Entwicklung eines Systems für generatives Sequenzieren und die Beschreibung einiger praktischer Beispiele unter Verwendung dieses Setups.

Die Einleitung fragt nach der Genese von Struktur und Bedeutung in der menschlichen Wahrnehmung und im Bewusstsein durch Gruppierungsprinzipien, um daraus einige grundlegende Faktoren der Wahrnehmung und Erfahrung hörbarer Strukturen abzuleiten. Außerdem wird ein kurzer thematischer und historischer Überblick über musikalische Sequenzierung und die Anwendung einiger generativer Methoden in Kunst und Design gegeben. Hinzu kommt eine kurze Einführung in die Software Puredata, wie sie im weiteren Verlauf dieser Arbeit zum Einsatz kommen wird.

Der Hauptteil der vorliegenden Arbeit konzentriert sich auf die Beschreibung eines generativen Sequenzierungssystems, das in seinen wesentlichen Komponenten beschrieben werden soll. Das generative Setup wird dafür in seiner Struktur und Funktionalität skizziert. Auf dieser Grundlage wird die praktische Umsetzung der verschiedenen Komponenten in Puredata beschrieben. Es werden einige Anwendungsbeispiele vorgestellt, die das hier entwickelte System für generatives Sequenzieren illustrieren. Dabei werde ich mich auf jene Aspekte konzentrieren, die für das Verständnis des Einsatzes verschiedener generativer Modelle und Methoden in der Praxis des Sound-Designs relevant sind.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	On the concept of structure	2
1.2	Musical sequencers: A short history of loop-based sound machines	15
1.3	The generative approach	20
1.4	Sequencing in Puredata (Pd)	28
<b>2</b>	<b>A generative sequencing system for algorithmic improvisation and its implementation in Puredata</b>	<b>33</b>
2.0.1	Algorithmic improvisation	34
2.0.2	A graphical model of a generative sequencing system	36
<b>2.1</b>	<b>The timing: A basic pulse as a grid for variable loops</b>	<b>39</b>
2.1.1	Pulses, beats and bars	40
2.1.2	Timing implementation in Puredata	41
<b>2.2</b>	<b>The communication medium: connecting user interface, structure generators and generative sound sources</b>	<b>42</b>
2.2.1	Communicating finite-state machines: The 'local states'	43
2.2.2	The grouping of play-modes: building systems, networks and hierarchies	44
2.2.3	Implementation in Puredata: The 'losta' object	45
<b>2.3</b>	<b>Structure generators</b>	<b>47</b>
2.3.1	Small-scale and large-scale structure	48
2.3.2	Synchronous and diachronous structure	49
2.3.3	Implementation examples in Puredata	52

<b>2.4.</b>	<b>The graphical user interface (GUI)</b>	<b>59</b>
2.4.1	The mixing interface	61
2.4.2	A GUI for improvisation with local states and structure generators	63
<b>2.5</b>	<b>The generative sound sources</b>	<b>65</b>
2.5.1	The 'Pitch Group Sequencer' (PGS): A two-dimensional probabilistic sequencer	66
2.5.2	The 'Loop Slice Mangler' (LSM): A looper and beat-slicer for generative manipulation of sound loops	73
<b>3</b>	<b>Conclusion and outlook</b>	<b>80</b>
	<b>References</b>	<b>82</b>

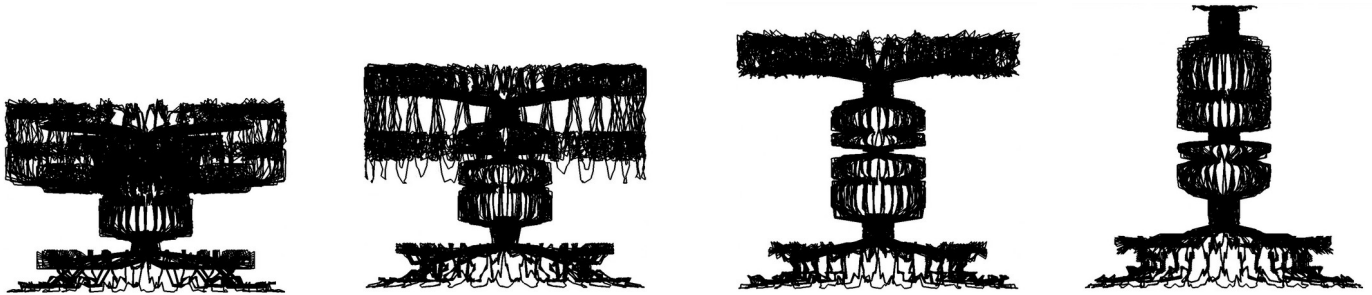
# Chapter 1

## Introduction

This thesis attempts to conceptualize and describe a setup for generative sequencing as I developed it out of practical implementations in the course of the design and realization of generative sound installations in the past few years. I am interested in generative techniques not only in the context of sound design and digital art but also in other areas of design, art, and craftsmanship. The increasingly easy availability of computers in recent decades made it an accessible tool for the implementation of generative approaches in art and design. But generative techniques and processes to produce artifacts should not be seen exclusively as a consequence of computerization. Many methods of producing objects and shaping the world (through farming, building, etc.) have presumably developed from natural formation processes that can be understood and described as generative in their nature. Iterative actions carried out repeatedly in time lead to the formation of structured objects (for example growth of living organisms, mineralization, and geological formation processes, settlement formation, etc). The computer makes it definitely easier to implement such generative processes and to display the results on the screen or in an audible form as sound. This is why I began to investigate the possibilities of building generative systems in a computerized form that provide a more complex and at the same time more organic way of arranging and structuring materials or elements in time and space to form a work of art.

Chapter 1 intends to introduce the idea of generative sequencing by providing a thematic and historical overview of the most important topics and concepts used in this thesis. The second and main part describes the generative sequencing system I have developed in the past few years in the field of sound design.

## 1.1 On the concept of structure



*Fig. 1 Evolving structure generated from an L-System interfering with sinusoidal waves mapped on a stereo-oscilloscope.<sup>1</sup>*

Observing the use of the word 'structure' it becomes clear, that the concept is suitable to describe a vast quantity of very different phenomena in all domains of human society and nature. It is employed generously when speaking about 'cultural' systems of human society but also when describing 'natural' phenomena and processes. 'Material structures include man-made objects such as buildings and machines and natural objects such as biological organisms, minerals, and chemicals. Abstract structures include data structures in computer science and musical form'<sup>2</sup>, states Wikipedia.

Apparently, the concept of 'structure' is very versatile but somehow ambiguous and confusing. A thing that one might notice from the beginning is, that structure is at the same time something attributed to objects of the material world and to mental or cultural systems, which have no clear material manifestation or existence. This double nature of structure is what makes it so funny and fertile to deal with it. When speaking of 'structure' it is not clear, if one is describing a phenomenon of human imagination and conscience or material reality. The double nature also points to something else: Structure is where the inner world of a self-conscious system like a human being (conscience or mind) meets and interacts with an external world (material reality). Perceived material structures manifest themselves in the 'inner world' of consciousness. The structuring of sensual perception allows the formation of a mental representation of the material world, a (subjective) 'reality'. This process of formation of an 'inner world' is the base on which a self-conscious system can define itself as a subject in opposition to an external world of objects.

On the one hand, structure connects the self-conscious subject to the world of objects. Perception and comprehension identify structures in the outside world to form a comprehensive (or useful and kind of 'well-structured') reality within the conscience of the subject. Without the ability to perceive

---

<sup>1</sup> Figures and illustrations for which no source is indicated are by the author.

<sup>2</sup> <https://en.wikipedia.org/wiki/Structure> (Jan 2021).



structures – and at the same time to structure perceived impressions and stimuli – the formation of a coherent conception of reality would not be possible.

And on the other hand, with every conscious action of a subject, mental structures are projected from its inner world into the material world. Doing or making something always means creating or unfolding a mental structure in the external world. In other words: Structures can be seen as a tool of self-conscious systems unfolding a productive or explicative power by connecting virtuality and material reality via a complex system of relations. In this thesis, I look more at the productive side of structure when talking about methods for generating content for generative sequencing systems.

The French philosopher Merleau-Ponty emphasizes the fundamental role of structure for the emergence of reality as comprehensive mental representation when describing structure as 'the joining of an idea and an existence which are indiscernible, the contingent arrangement by which materials begin to have meaning in our presence, intelligibility in the nascent state.' (Merleau-Ponty 1963, 206)<sup>3</sup>

Speaking about sequencing systems and the generation of sequences involves a concept of organization or structure, where distinct elements are arranged in time (or space) to form some kind of perceivable order or system. As an artist and designer, I am first of all concerned by the aesthetic implications of the underlying concept of structure.

But can we define the meaning of structure more precisely? And what about the perception of structure? Are there general principles that let us perceive something as 'structured', or 'structure' itself? Why is a certain structure perceived as interesting or beautiful?<sup>4</sup> Does the perception of beauty depend on the cultural reference system of the spectator or are there other structuring mechanisms in human perception that we should consider when designing a structured work of art?

To shed light on these questions, I would like to take a closer look at the concept of 'structure' and different aspects of its meaning. Firstly, I want to check some lexical definitions and the etymology of the word structure. Secondly, I will focus on the perception of structure and the question of how perceivable structures emerge in human consciousness. Thirdly, I will try to draw some conclusions on the production of structures by sequencing sound – as the main question of this thesis is to figure out, how to design a system for structuring sound events in time.

---

3 The French original of this quotation is: 'Ce qu'il y a de profond dans la « Gestalt » d'où nous sommes partis, ce n'est pas l'idée de signification, mais celle de structure, la jonction d'une idée et d'une existence indiscernables, l'arrangement contingent par lequel les matériaux se mettent devant nous à avoir un sens, l'intelligibilité à l'état naissant.' Notice that Merleau-Ponty relates his consideration on the concept of 'Gestalt' that we will see later in this chapter.

4 A reason could be its particular outstanding strangeness, newness, or funniness. Another very different reason could be that it seems to be a 'true' embodiment of what we feel to be our 'natural', 'harmonic' core, or truth. These two examples show the two extreme positions of an experimental approach and a classical, universalist approach. For a detailed discussion of this Topic see (Eco 2002, 378f).

## Structure: Lexical definition and etymology

A definition from the Cambridge Dictionary of English is the following: 'structure (arrangement): the way in which the parts of a system or object are arranged or organized, or a system arranged in this way.'<sup>5</sup>

An etymological definition: 'structure (n.): mid-15c., "action or process of building or construction;" 1610s, "that which is constructed, a building or edifice;" from Latin *structura* "a fitting together, adjustment; a building, mode of building;" figuratively, "arrangement, order," from *structus*, past participle of *struere* "to pile, place together, heap up; build, assemble, arrange, make by joining together."<sup>6</sup>

Another definition from Wolfhart Henckmann in his *Encyclopedia of Aesthetics* is as follows: structure is 'a set of relations by which the elements of a whole are connected to each other' (Henckmann/Lotter 2004, 350). The author gives also some helpful further specifications:

- The whole has logical precedence over its elements.
- The elements are not to be determined in isolation from each other, but by their function in the whole.
- The relations that exist between the elements are not to be understood as monocausal, but as reciprocal relationships.
- A structure tends to behave invariantly in its internal regularity in the case of changes in individual elements or external influences.

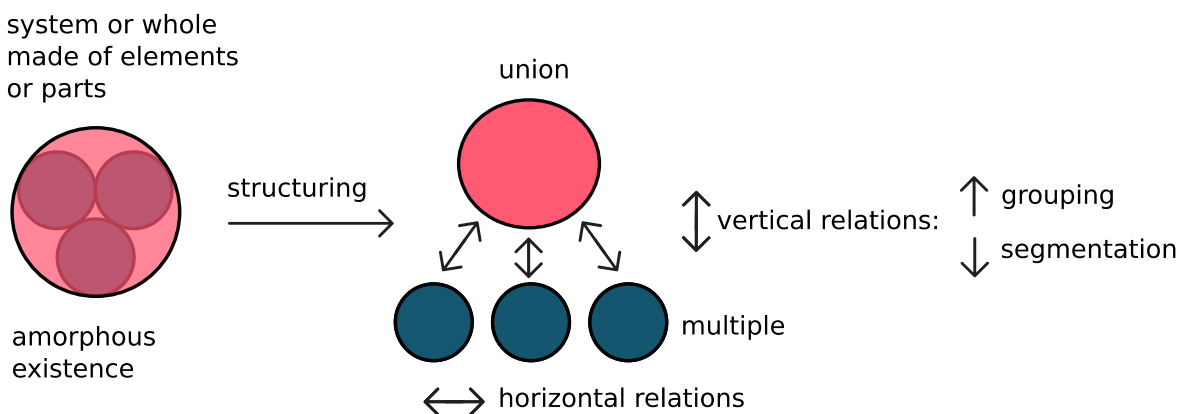
---

5 <https://dictionary.cambridge.org/dictionary/english/structure> (Jan 2021).

6 <https://www.etymonline.com/word/structure> (Jan 2021).

## Grouping and segmentation

It follows from these definitions that the central mechanisms (for material objects) or mental faculties (for cultural systems) of structuring concern the arrangement and interrelation of various objects that form a whole. Structuring thus means relating objects on different scales by arranging them – either by grouping several elements into some kind of unit, or by segmenting or differentiating a unit into multiple elements. Grouping<sup>7</sup> or segmentation introduces vertical or hierarchical relations into a multiplicity of elements. Based on this process we can perceive and create structures in manifold ways from the most basic or obvious, to the most complicated and unexpected grouping and segmentation methods. A 'basic' method would be grouping several objects together because they have the same color. A more complicated or 'unexpected' method would be, for example, to use a microscope for differentiating functional cellular systems within living tissue.



**Fig. 2** Basic structure model for introducing vertical and horizontal relations to distinguish the elements of a whole.

It follows from these considerations that we can understand a multiplicity of things having a 'surface', where the individual elements show up, and a 'depth', where the systemic relations come to light. The interplay of these two levels gives rise to 'structures', i.e. unities or wholes, consisting of various elements characterized by a group relation<sup>8</sup>. So we can assume, that in a structured sequence of events or objects there is some kind of 'higher' or 'deeper' level that expresses itself in the sequence seen on the surface. If there are various coordinated levels of organization on different scales present in an object, it appears to be a structured system. Such systems act in manifold ways in the perception and interpretation of phenomena as well as in the generation of structured content.

7 Grouping is described as a basic structuring mechanism in music perception by Lerdahl and Jackendoff (Lerdahl/ Jackendoff 1983). The authors consider grouping analysis to be the most fundamental component of musical understanding. It articulates a hierarchical segmentation of a musical piece into phrases, motives, periods, and sections of different lengths.

8 Cambouropoulos emphasizes the importance of such structured unities, which already consist of several elements, for musical perception. 'There is evidence that things such as melodic and harmonic pitch intervals, chords or larger configurations such as tone clusters, tremolos, trills, glissandi are commonly perceived by listeners as wholes rather than combinations of atomic lower-level components.' (Cambouropoulos 2010).

## Language as a structured system

In the study of philosophy, the natural sciences, and the humanities in the 20th century, the concept of structure is very present and is used in many different areas and fields of study. A philosophical and humanistic movement called 'structuralism' (and later 'poststructuralism') has influenced different areas of natural and social sciences. Mostly these structuralist approaches use concepts of structuralist linguistics in the succession of Ferdinand de Saussure to describe different social phenomena as structured systems in analogy to language<sup>9</sup>. Where language is understood as a structured system that uses elements (signs or words) according to grammatical rules. De Saussure brought up some influential concepts about language and signs, the most important being the following:

- The opposition of 'language' as a system versus 'speech' as a practical realization of the language.
- The concept of a sign, being composed of a 'signifier' (i.e. the material sign) and a 'signified' (i.e. the meaning).
- The arbitrariness of the relation 'signifier' – 'signified' (i.e. the word is not bounded to its meaning by a natural relation but a convention).
- The signs gain their meaning from their relationships and contrast with other signs, not because of their reference to a signified object.<sup>10</sup>

These assumptions lead to a concept of language as a system that does not simply describe or represent the world of phenomena. Language functions according to its internal structure. It doesn't even require an external world to be a functioning system, it only needs a set of internal relationships<sup>11</sup>. This linguistic view of system and structure can be applied to other cultural systems. The elements of a system then appear as signs having a material side ('signifier') and a relational side ('signified' or meaning), even if they are not part of what is traditionally is called a language. Structuralism tries to expand this method of analysis to different cultural systems and certainly had a great influence on the concept of 'structure' and 'system' in the 20th century. The ontological content of the 'signified' (i.e. the 'meaning' of a sign) in the structuralist model has been the object of interesting discussions in philosophy, semiology, and the social sciences. Anyway, structuralist approaches would explicate the 'signified' or meaning as a result of structure: The positions and relations within a system define the meaning of a thing, whereby language is seen as a model for other cultural systems. When considering musical traditions that use symbolic notation systems and

---

9 The most influential structuralist thinkers are Levi-Strauss in Anthropology, Jaques Lacan and Jean Piaget in psychology, Roland Barthes in literary theory, Louis Althusser in sociology. The idea of using language and its structure as a model for the analysis of different areas of human culture is one of the most influential epistemological concepts of the 20th century. The widespread use of this approach is also called the 'linguistic turn'. (Münker/Roesler 2000, 19).

10 For detailed information on that subject see (Saussure 1995), (Eco 2002, 62f), (Münker/Roesler 2000, 1f).

11 De Saussure spoke about 'syntagmatic' and 'paradigmatic' relations. 'Syntagmatic' is a relation of succession (within a sentence) and 'paradigmatic' a relation of possible substitution.

certain (harmonic) rules about the use of notes within a sequence, it seems promising to analyze this system in analogy to language<sup>12</sup>.

A theory somehow based on a structuralist concept of language is the approach of generative grammar brought up by Noam Chomsky in the 1950s. This theory allows a more accurate analysis of syntactic rules that form the sentences of a speech<sup>13</sup> (while the 'paradigmatic' and 'syntagmatic' axis shown by de Saussure seem to be a very general characteristic of the relations of the signs within a language and do not give exact information about how to build correct expressions). Chomsky also differentiates a 'surface structure' and 'deep structure', where the 'surface structure' is one possible and syntactically correct version of a 'deep structure' that represents the semantic content of the sentence<sup>14</sup>.

Without going into further detail on linguistic theories and structuralism, their ontological implications<sup>15</sup>, and their big influence on various areas (from philosophy to automata theory and theoretical informatics), I would like to note the value of these concepts for generative methods working on a symbolic level. Such methods will be discussed later in this thesis.

## Perceiving structures

Being an artist and designer, the concepts of structure and system are very important when deciding how to design the parts or details of a work in relation to each other and to the whole. A work of art can be assembled from the same elements or materials, but using different arrangements will result in completely different pieces, and will produce very different feelings, ideas, and impressions in the audience. In such a case the perceived difference clearly results from the different relations that are exposed or expressed by the whole piece of art, not from the materials or elements which were used. So if 'structure' concerns the relations of the parts of an organized wholeness or system<sup>16</sup>, it is certainly one of the main aspects to consider when designing a work of art.

In any case, the structures that are considered valuable in a piece of art can be very different. They can concern the perfect material realization of complex patterns or the position a work occupies within an aesthetic or historical discourse. This example also shows that a structure, that is seen or interpreted within a work of art, depends on both, the perception and the interpretation of the creator or spectator. But structure is not something that is there in the phenomena of the world and can

---

12 See (Eco 2002, 107, 382f).

13 See (Nierhaus 2008, 83f) and chapter 1.2 of this thesis.

14 See (Chomsky, 1964). In this work, Chomsky first uses the distinction between deep structure and surface structure.

15 The ontological question is, what the 'structures' identified in different systems exactly are. It is not clear if they are constructed in human conscience or if they are something real inherent to the material world. It is equally unclear whether it is possible and reasonable to seek or presuppose general basic structures of the human mind such as a 'universal grammar'. In the production or analysis of structures, however, it makes quite a difference whether we intend to express a universal code or whether we use structures, grammar, signs, and reference systems only as tools to create structures. Following Umberto Eco, 'serial thinking' concerns the production of variable structures without reference to a universal basis. This is opposed to 'structural thinking', which tries to uncover the underlying basic structures of a system through structural analysis (Eco 2002, 387f).

16 See (Henckmann/Wolfhart 2004, 352).

simply be perceived by everybody the same way. Relations and arrangements to some extent only exist when perceived, understood, or constructed as such in an observer's mind.

Considering the great amount of different social structures, languages, and cultural systems, it is certainly an interesting task to search for structures that are universal to human conscience, like a 'universal grammar'<sup>17</sup> or invariant structures in cultural systems that would in some kind manifest invariant human brain structures like intended in some structuralist ethnology<sup>18</sup>. We can also observe a tendency to see one's own cultural group structures as a kind of normality, with a leaning towards universality and a devaluation of cultural systems originating from other ethnic groups – eurocentrism in science and global capitalism (the free market as a 'natural' and universal form of the economy) are both examples of this. In art and design, universalist approaches that emphasize the importance of a stable and unique reference system still have their justification as a basis of art history. Postmodern and poststructuralist theories have changed a lot in academic philosophy and the social sciences by breaking up universalist tendencies. Also in art and design, experimental use of structures, thinking of them as temporary and changeable is more accessible and accepted today<sup>19</sup>.

Disregarding the danger of universalist approaches I will try to identify some principles of human perception of material structures, asking for these structuring principles of the perceptual apparatus on a more psychological level<sup>20</sup>. Because if artists or designers think of their work of as a structure or a structured system, they should ask themselves, if and how an audience or an observer is able to perceive the intended structure. When working or designing with sequencers, the question is how to arrange different sound events to create perceivable structures in the mind of the listener. And furthermore:

- Which structures are easily perceivable and clearly understandable because they refer to a very broad, popular, or unquestioned perception scheme and/or interpretation system?
- Which ones are freer, more ambiguous, or controversial? These aspects change a lot, depending on the location where a work of art is shown and the interpretation system present in the observer's minds.
- Are there very general mechanisms in human perception, which serve us to structure the things we perceive – like a common ground of perceiving structures and/or structuring perception?

---

17 Chomsky argued that the human brain contains a limited set of constraints for the organization of language. This implies that all languages have a common structural basis, a sort of 'universal grammar'. See (Dąbrowska 2015) for a critical review of this concept.

18 See (Münker/Roesler 2000, 9f).

19 For a detailed discussion of the different views of structure and the resulting ontological and aesthetic implications see (Eco 2002, 378f). Eco contrasts 'structural thinking' with 'serial thinking'. The former tries to find universal structures or codes, while the latter is based on the idea, that every message questions and reformulates the code and the structure itself.

20 To describe the structuring function of human perception in sound processing: 'Our perceptual mechanisms actively try to organize the infinite variety and nuances of an input musical signal into manageable perceptual events.[...] The elementary events perceived as constituent units of an acoustic continuum are further grouped together into elementary categories.' (Cambouropoulos 2010, 131).

When we perceive the existence of something – one or multiple objects (amorphous existence in figure 2) we can introduce structure either by differentiating or splitting elements within the existing thing(s) or element(s) or grouping or parenting various things or elements to a superordinate entity.

Difference and union are what I called vertical relation in figure 2 – splitting up something into its elements or joining the elements to one superordinate (conceptual or material) entity. The question that follows is why we would perceive a unity or difference in a set of perceived objects. Intuitively I would suggest that the presence or absence of the following factors is decisive: proximity, similarity, continuity/row/evolution, repetition, simplicity/clearness/closeness. This basic grouping and differentiating mechanisms are applied both in perception and production of structure (as the double nature of structure supposes).

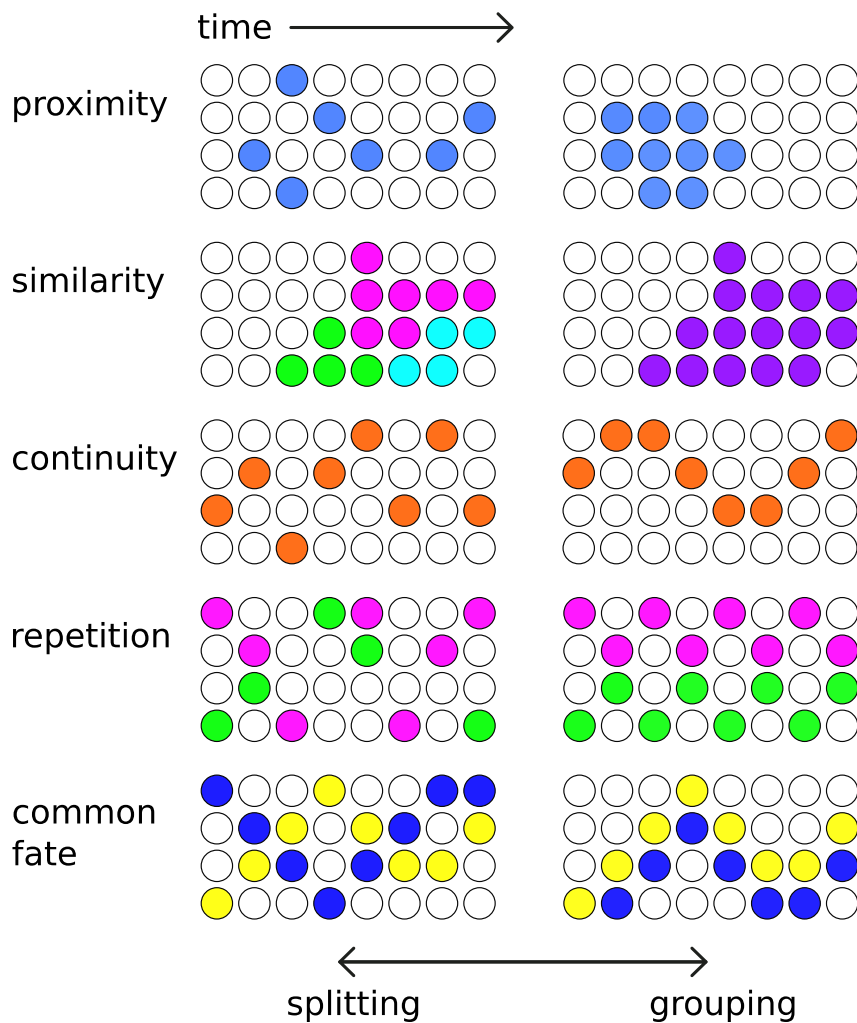
I found some similarities to this approach in Gestalt-psychology, where the Austrian psychologist Max Wertheimer formulated six essential factors for the formation of coherent arrangements to form a 'Gestalt' or shape in human perception: 'proximity, similarity, simple shape, continuity, closeness, and common fate' (Wertheimer 1923). Wertheimer listed the factors that would influence the perception of wholeness, which he called 'Gestalt', mostly translated as shape. A 'Gestalt' is made of various elements in a group relation. Similar to what I would expect to be a grouping mechanism in a structure model. It seems that Gestalt-psychology considers more the visual and spatial part of human perception (at least regarding the examples mostly cited when explaining the factors or rules). But also in audible experience and the techniques of sequencing 'proximity', 'similarity', 'continuity' can be easily identified as important factors for perceiving and producing audible structure. Also 'simple form', 'closeness', and 'common fate' can in certain ways be applied to audible structures<sup>21</sup>.

To illustrate this fact, I transferred the most basic factors of structure perception to a graphic model (figure 3) showing time slots and events in a grid – as we are used to in musical sequencing. The model shows basic structuring mechanisms in sound perception for grouping several sound events or objects<sup>22</sup> into one audible object. The horizontal axis represents time and the vertical axis an audible property (it is very common to put the pitch on the vertical axis like in traditional European notation but one could use other audio properties, like volume, timbre, waveform, reverb, filter, etc.). The different colors represent different audio events or samples used for sequencing. An audible object can be a single event or more likely a sound stream (i.e. a perceivable sequence in time) constructed by arranging sound events of appropriate properties in a relationship of temporal proximity or regularity.

---

21 Lerdahl and Jackendoff investigated this question in their 'Generative Theory of Tonal Music'. Lerdahl: 'Investigation of the grouping component brought us to Gestalt psychology, for which there was an extensive literature on visual grouping. It was unclear how to make a grammar out of the Gestalt principles of proximity and similarity. Working through many grouping and metrical analyses, we found that the phenomena were gradient rather than categorical.' (Lerdahl 2009, 189). See also (Lerdahl/Jackendoff, 1983).

22 A sound event can be understood in analogy to Pierre Schaeffer's 'sound object' which according to him is the basic unit of sound, comparable to a unit of breath, articulation, or instrumental gesture. Schaeffer also speaks of an 'acoustic action' and an 'intention of listening' reflecting the two sides of identifying a sound event: the production and the perception: 'L'objet sonore est à la rencontre d'une action acoustique et d'une intention d'écoute.' (Schaeffer 1996).



**Fig. 3** *Creating audible structures: Basic mechanisms of grouping or differentiating sound events in time.*

The most decisive factors for audible structure perception considered here, are the following:

- **Proximity** in time and quality: If various sounds are heard within a shorter period of time and if their perceivable qualities are closer (sounding more similar in certain aspects) it is more likely, that they are perceived (and interpreted within a larger system) as one audible object.
- **Similarity** in time and quality: If different sounds in a period of time and range of quality are replaced by similar sounds they are perceived more likely as one audible object<sup>23</sup>.

23 'Without similarity, music would not be possible, would not exist. similarity, and its counterpart dissimilarity or difference, enables a listener to break down the acoustic continuum into smaller constituent parts (such as elementary events, segments, groups, streams), and to make associations between these parts (repetitions, variations, oppositions, transitions and so on). Local similarities and discontinuities/dissimilarities give rise to elementary discrete entities such as notes, and allow the formation of musical streams (e.g. voices) and segments within streams.' (Cambouropoulos 2009, 7).



- **Continuity** in time and quality: If a sound is repeated in time with a changing audible quality, it is more likely to perceive one audible object, if the time interval is constant or regular and the change in quality is continuous.
- **Repetition** in time and quality: If a combination of various sounds is repeated in time constantly it is more likely perceived as one audible object. Repetition can take place on different timescales and is related to the fact that perception and cognition are learning and remembering systems. It will more likely interpret arrangements as a group if perceived several times before<sup>24</sup>.
- **Common evolution or parallelism** in time and quality: If various sounds are changing in time (evolving) in parallel (concerning a certain audible quality) they are more likely perceived as one audible object.

---

24 The recognition of formerly experienced entities when grouping and structuring perceived elements of music or text is an important factor of structure perception. For that reason, Rens Bod argues for using a data-orientated parsing model combining the 'likelihood principle' (based on the recognition of repeated structures) in combination with a 'simplicity principle' (based on the recognition of the simple structures in the tradition of Gestalt-recognition). 'Our key idea is that the perceptual system strives for the simplest structure (the "simplicity principle"), but in doing so it is biased by the likelihood of previous structures (the "likelihood principle").' (Bod 2002, 289).

## Musical streams

Note that an audible object is not meant to be the complete fusion of various sound events into only one perceivable object. As shown in figure 2 it is a matter of structuring a set of objects with the help of vertical relations (i.e. grouping or segmenting). The grouping of several sound events in our perception and experience allows us to perceive a multitude of objects as a unity or a whole. But since sound is always a temporal and transient phenomenon, one must think of this unity more as a *stream* than a stable object. A musical piece can be imagined as various streams of sound in relation to each other and the whole piece. A structured unity is not created by fusion, but rather by keeping the inner multiplicity. The mechanisms illustrated in figure 3 are reversible in the sense that it is possible to group several objects to one and also split up a composite object into its elements. A stream is characterized by both, continuity and discontinuity, unity and multiplicity.

From the basic musical structuring mechanisms identified in this section, one can imagine the emergence of more complex musical structures – like streams or voices – that develop a kind of interaction, play, and meaning in a musical piece. With the help of principles like similarity, proximity, repetition, and variation it is to a certain point possible, to explain this emergence<sup>25</sup>.

A musical stream or sound stream will be formed when certain sequences or variations of sound events are placed and perceived in temporal continuity or parallel movement<sup>26</sup>. In the practical work of producing sound with various sequencers, the handling of musical streams is decisive. According to (Cambouropoulos 2008, 83) five principles are crucial for the experience of a musical stream within the listener:

- Principle of Temporal Continuity
- Principle of Tonal Fusion
- Pitch Proximity Principle
- Pitch Co-modulation Principle
- Onset Synchrony Principle

---

25 Cambouropoulos describes the emergence of musical entities from these principles descriptively: 'Similarity is at the very heart of music because it enables the emergence of the core musical entities themselves (for instance, a musical theme is not an object/event that exists out there in the world — it exists primarily by virtue of self-reference via repetition and variation).' (Cambouropoulos 2009, 8).

26 'Listeners break down the acoustic continuum into musical streams; they perceive streams of musical events such as streams of notes (e.g., melodic lines) or streams of chords (e.g., accompaniment). A number of perceptual factors [...] enable a listener to integrate or fuse co-modulating components (e.g., partials or notes moving in parallel) into coherent events and sequences of events, and, at the same time, to segregate them from other independent musical sequences.' (Cambouropoulos 2010 141). For a detailed analysis of voice and stream formation and segregation in human perception see also (Cambouropoulos 2008).

## Timescales and metric relations

Another important aspect when dealing with structures in music or sound art in a wider sense is the question of timescales and metric relations. As argued above continuity or discontinuity in metric aspects are important grouping relations in audible perception. The factors illustrated in figure 3 are at a first view adapted to sequencing sound in relatively short loops – like in traditional musical sequencers. But when working on larger pieces the question of sequencing on a larger timescale becomes more relevant. Especially when trying to build a system that is generative in some way not only on a short timescale of pattern production but also on the larger timescale of creating parts of a piece by generating large-scale structures. The large-scale segmentation organizes the small-scale structures in some way<sup>27</sup> – like in a time-based tree model. When describing my practical setup for generative sequencing in chapter two I will present a model that takes this fact into account.

Musical timescales can be described as logarithmic scales, as the doubling is a more accurate way to relate different time intervals than linear growth. This fact is taken into account in the definition of semitones as the twelfth of a doubling or halving. The concept of a logarithmic timescale can also be employed for the description of larger periods, like beats, bars, or functional parts of a musical piece.<sup>28</sup> Combining different timescales results in a tree model where events belonging to a shorter timescale can be grouped into an event on a larger timescale, while large timescale events can be split up into various small timescale events. The presence of various interconnected timescales in a musical piece is an important structuring mechanism for creating structural depth – both in perception and generation of structures. In this sense, it also is important to pay attention to the timescales when working with structure principles like similarity and repetition in practice, regarding attention span and short-term memory of listeners.

---

27 Lerdahl and Jackendorf propose 'time-span reductions' and 'prologational reduction' for the hierarchical structuring on different timescales based on information obtained from metric and grouping structures. They form tree-like hierarchical structures that combine time spans at all temporal scales of a piece. See (Lerdahl/Jackendoff 1983, 119f) and (Lehrdal 2009).

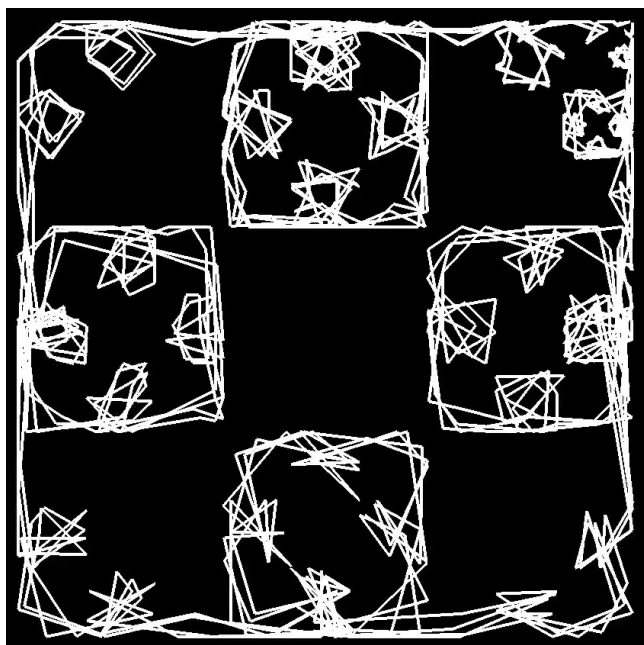
28 Essl speaks about 'micro-time' and 'macro-time' relating timbre and tonal qualities with rhythm and form. 'Die gleichen Prinzipien, die im Bereich der Tonhöhen ("Micro-Zeit") gelten, haben auch auf dem Gebiet der Dauern ("Macro-Zeit") ihre Berechtigung. So lassen sich - ebenso wie Tonhöhen - auch Dauernwerte in Zeitoktaven zusammenfassen.' (Essl 1996, 19).

## Structure generators

So how can a structure be generated or produced in the field of sound design? If we take the above consideration and define structure as the way how parts are organized to form a whole or a system, this means that we have to find ways to arrange and relate the elements to form a more complex entity. Such a structuring process can happen in a lot of different ways depending on the approach used to interpret and/or generate an arrangement or system. In chapter 1.3. I will describe some generative approaches that can be used as a methodological orientation for algorithmic implementation.

I found the term 'structure generator' in the sense of a machine creating structures for musical sequencing in a work by Karlheinz Essl (Essl 1996). He speaks of 'structure generators' in the sense of theoretic objects, that can be implemented in a software<sup>29</sup>. This software is able to map structure to sound parameters and to produce the respective sound. I will use the term structure generator in this sense in chapter 2 when describing the design of a generative sequencing system.

Another important quality that structure generators should have is the ability to connect on different levels of control in terms of timing or other functional relations. In this way, it is possible to create generative control networks. A structure generator can be controlled by another structure generator from a different place and/or hierarchy in the structure. In this sense, structure generators are like pattern generators including depth.



**Fig. 4** A Koch curve<sup>30</sup> generated with an L-system mapped to a stereo oscilloscope in Pure Data.

29 Essl already used a visual dataflow programming language, 'MAX', similar to Puredata in many functional aspects, in the 1990s. See (Essl 1996, 27).

30 See (Lindenmayer/Prusinkiewicz 2004, 10). The curve can be drawn by applying a 'turtle interpretation of strings' generated by a Lindemayer system.

## 1.2 Musical Sequencers: A short history of loop-based sound machines

### The subdivision of time

A sequencer is a concept, apparatus, or machine that divides time into sections that form a sequence. The different sections can be assigned or related to an event, a concept, or an activity. In this general form, a wide variety of 'sequencers' can be found in the history of mankind and their origins go back to prehistoric times. Many archaeological findings suggest prehistoric concepts of time-division: The 'Lebombo Bone', the 'Blanchard Bone' or the 'Ishango bone' are the most famous examples of paleolithic tally sticks dating from around 40.000-30.000 years ago. These objects were probably created to conceptualize and organize time and are therefore the oldest-known ancestors of the calendars and clocks we use today.

In the history of music, the conceptualization and division of time is obviously of fundamental importance. The drawing or writing of music<sup>31</sup> presupposes such a conceptualization of time and periods or segments of time.

However, music and musical traditions require the sequencing of time, also if they are not written down. It can be assumed that musical practice is just as much a causal factor in the development of an abstract concept of time and the division of time as the conceptualization of time enables more complex musical structures, traditions and works.

### Repeating patterns

The idea to combine a temporal grid with sound events is the basis of the musical sequencer<sup>32</sup>. It enables the production of patterns and their repetition – which is an essential creative element of many musical styles.

'The idea of a grid has been one of the most prevalent characteristics of music throughout the past few centuries. The reliance on a sonic grid with repeating rhythmic and melodic motives has become imbued into the human ear, and the sequencer in its many forms has become a popular interface for music creation and sound music computing.' (Arar/Kapur 2013, 384)

Musical sequencers in the narrower sense are tools that externalize the division of time and the connection of certain points in time with sound events from human consciousness into the material world of objects. For this purpose, it is necessary to design apparatuses that are able to perform this task. In the following, I will give a brief historical overview starting with the first mechanical sequencers.

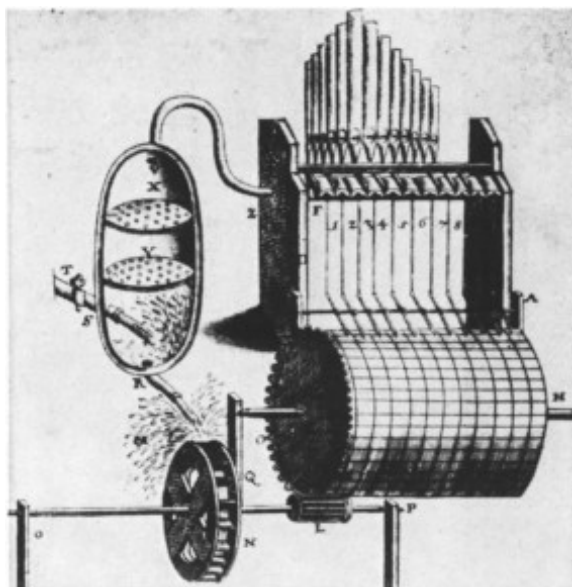
---

31 The oldest record of musical notation is a cuneiform tablet from Babylon dating from around 1400 BC.

32 See (Davis 2001).

## Mechanical sequencers

The first known music sequencers are mechanical in nature and work by means of a rotating cylinder with pins that trigger different sounds. The first music automaton to function in this way was a hydro-powered barrel organ described by the Persian Banū Mūsā brothers in the year 850 in their work entitled 'Book of Ingenious Devices'. The concept of a cylinder with raised pins on its surface remained the basic device for the mechanical production and reproduction of music until the second half of the nineteenth century.<sup>33</sup>



**Fig. 5** Drawing of the water organ at the Villa d'este, Rome, 16th century (Fowler 1967, p. 46).



**Fig. 6** The barrel organ 'Salzburger Stier' from 1502<sup>34</sup>.

A famous example of an early barrel organ is the 'Salzburger Stier' (figure 5), dating from 1502 (commissioned by Archbishop Leonhard von Keutschach). It is still in use today at the castle of Salzburg (Austria). The development and refinement of barrel organs led to the implementation of barrel pianos, which combined a rotating cylinder (the 'barrel') with pins with the triggering of the hammers of a piano. The refinement of these instruments culminated in the late 19th century in the production of the so-called 'Player Piano' or 'Pianola', a self-playing piano that most commonly used punch-cards (perforated paper) to automatically trigger the hammers of the piano. The first practical automatic piano was invented in 1896 by Edwin S. Votey and called the 'Pianola'.

Towards the end of the Middle Ages, the craft of watchmaking developed further in various parts of Europe, making it possible to build more sophisticated automatic musical instruments leading to

<sup>33</sup> See (Fowler 1967, 49).

<sup>34</sup> Source: [www.salzburg-rundgang.at](http://www.salzburg-rundgang.at) (Jan 2021).

the industrial production of 'musical boxes'. The first music box factory was opened in Sainte-Croix in Switzerland in 1815 by Jérémie Recordon and Samuel Junod.

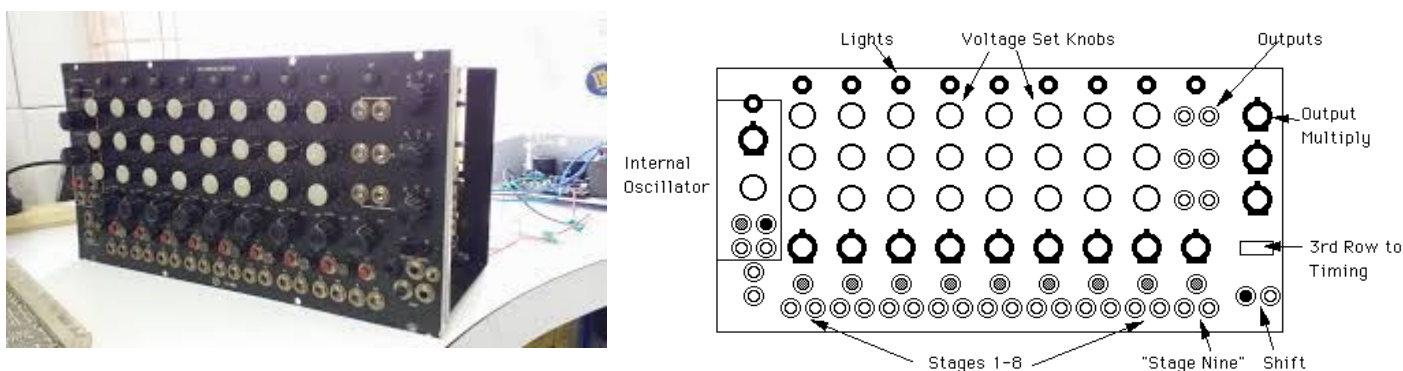
During the Baroque period, a wide variety of musical automata were elaborated –including diverse mechanisms for the sequencing of sound events. A famous example is the 'Flute Player' from 1737 designed by the French inventor Jacques de Vaucanson.

In 1877 Thomas Edison invented the 'Phonograph', which had a big influence on the development of music reproduction machines. It combined the physical principles of the turning cylinder with the recording and reproduction of sound waves.

## Drum machines and analog sequencers

The first electronic drum machine was the 'Rhythmicon' which was invented in 1931 by Henry Cowell, an American composer, and musical theorist, in collaboration with the Russian inventor Léon Theremin. In the 1950s and 60s other drum machines were designed by Harry Chamberlin and the Wurlitzer Company.

Another field where sequencing mechanisms were developed and used is the design of analog synthesizers. In 1957, Herbert Belar and Harry Olsen developed the RCA Mark II Sound Synthesizer at the Electronic Music Center of Columbia-Princeton University which is considered the first analog sequencer. On this basis, Robert Moog developed the 'Moog 960' in 1968. These early sequencers were based on voltage-controlled processes where parameters such as frequency, duration, modulation or filter characteristics could be set with a knob for each step.<sup>35</sup>



**Fig. 7** The Moog 960<sup>36</sup>.

A further important figure in synthesizer and drum-machine development was the American composer and inventor Raymond Scott, who worked on electronic devices for sequencing sound

<sup>35</sup> See (Arar/Kapur 2013, 384).

<sup>36</sup> Source: <http://artsites.ucsc.edu/EMS/music/equipment/synthesizers/analog/moog/Moog.html> (Jan 2021).

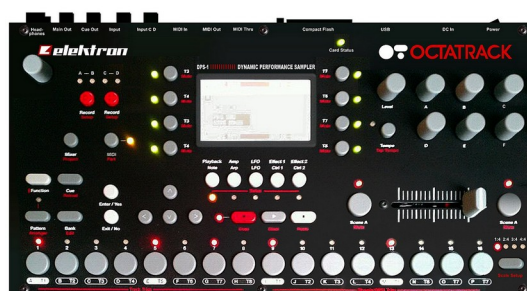
from the 1960s onwards. Until the end of his life in 1994, he kept developing the 'Electronium' a combined electronic synthesizer and algorithmic composition and generative music machine<sup>37</sup>.

In the 1960 and 70s, the progress in transistor technology and other fields of electrical engineering led to the development of programmable drum machines and the use of digital sampling instead of analog sound synthesis in the 1980s. The probably best-known drum machines from this decade are the 'Linn LM1' (1980) and from the Roland Cooperation the 'TR-808' (1980) and 'TR-909' (1983). All three were programmable, The 'Linn LM1' and the 'TR 909' already use digital sampling while the 'TR-808' still used analog synthesis for sound production.

Until today, the design of standalone sequencers and drum machines is mostly based on the idea of 8 or 16 steps with assigned sounds running in a loop<sup>38</sup>. A more recent example of a standalone sequencer is the 'Octatrack' made by 'Elektron Music Machines' in 2010. 'An LCD surrounded by hand controls let the user interact with sampler, step sequencer, mixer, and effect features while stereo audio and MIDI signals are continuously output.' (Ekelund/Mårtensson 2013)



**Fig. 8** The Roland TR-808 (1980)<sup>39</sup>.



**Fig. 9** The Elektron Octatrack (2010)<sup>40</sup>.

37 For more information on Raymond Scott's life and work see: <https://www.raymondscott.net> (Jan 2021).

38 See (Butler 2006, 46f).

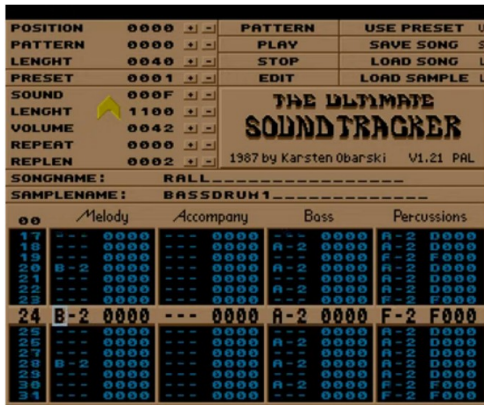
39 Source: <https://rolandcorp.com.au/blog/roland-drum-machine-chronicle-1964-2016> (Jan 2021). This webpage gives an historic overview over the drum-machines from the Roland company.

40 Source: <https://www.elektron.se/products/octatrack-mkii/> (Jan 2021).

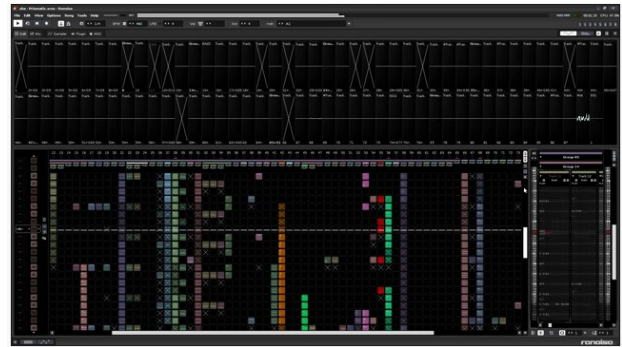


## Software sequencers

The first software sequencer was created as part of the 'ABLE' computer in 1975 by New England Digital. In the 80s and 90s, rapidly growing computerization led to the development and use of more software sequencers. Karsten Obarski wrote the first so-called 'tracker' software in 1987 for the Commodore Amiga which was called 'Ultimate Soundtracker'. It consisted of four parallel sequencers using 8-bit samples. The position in time is displayed numerically on the screen and shifts as a composition or track proceeds in time.<sup>41</sup> In the following decades other tracker software systems were released keeping essentially the vertical multi-track step sequencer design increasing the number of editable tracks and improving other features like sound quality, effects, etc.



**Fig. 10** The Ultimate Soundtracker 1987<sup>42</sup>.



**Fig. 11** The tracker software Renoise 3.2 from 2019<sup>43</sup>.

Today all common digital audio workstations (DAWs)<sup>44</sup> on the market include some kind of software sequencer often using MIDI data and virtual instruments in a timeline for creating patterns that can be repeated, modified, and arranged in a thousand ways.

41 For an overview see (Arar/Kapur 2013, 386).

42 Source: screenshot from <https://www.youtube.com/watch?v=5ywaCR5Tg4A> (Jan 2021).

43 Source: screenshot from <https://www.youtube.com/watch?v=1p5vAxnPGkQ> (Jan 2021).

44 To give some examples: Ableton Live, Logic Pro, Reaper, Pro Tools, Steinberg Cubase, etc.

## 1.3 The generative approach



*Fig. 12 Samba, a sculptural work by Guto Requena using methods of generative design<sup>45</sup>.*

### Generative Art

The basic idea of generative art or design is to use a set of rules or constraints to create a piece of work (artwork, design, text, etc.) out of some initial input through a partly or wholly automated production process. The set of rules employed determine the properties and features of the piece to be produced and is usually linked to a production mechanism that expresses these determinations in a perceptible form (graphics, sound, etc.). In many cases, a computer is used to execute a generative algorithm defined for this purpose. The result is then presented in a sensually perceptible form. Obviously, the mapping of the results of the generative algorithm into a material form is often decisive for the result. The reason, why the generative design approaches involving a computer have been used increasingly in recent decades, is the wide availability of sufficiently powerful computers. The computer nowadays is a very powerful tool to implement generative processes and show them in a sensually perceptible form. Often it even becomes a kind of co-creator<sup>46</sup> in the process of producing content with generative algorithms.

The fascination of generative methods for creating structures and patterns is partly because they derive complexity from often simple rules and present this complexity in a visually or audibly captivating way. The underlying mechanisms or algorithms are not obvious at first sight, therefore the result is often surprising and unpredictable. The use of randomization of production parameters and variables is often an important element in the generative process<sup>47</sup>.

For me, as an artist and designer, generative approaches are particularly interesting because they shift the role of the designer as a creative momentum: He or she no longer operates on the level of the realization of elaborate details, but on the level of creating some kind of generative plan, that

---

<sup>45</sup> Source: [https://en.wikipedia.org/wiki/Generative\\_design#/media/File:Samba\\_Collection.JPG](https://en.wikipedia.org/wiki/Generative_design#/media/File:Samba_Collection.JPG) (Jan 2021).

<sup>46</sup> See (Nierhaus 2021, 13).

<sup>47</sup> 'Finally, randomness is an essential component of most generative computer-based procedures, since the implementation of algorithms almost always defines a "stochastic scope" that allows the generation of different instances of a common structural idea.' (Nierhaus 2021, 12).

determines these details. In such a generative plan, which can be expressed as a computable algorithm, decisions about essential aspects of the work are partly taken over by the corresponding algorithm<sup>48</sup>. This creates the fascinating impression that structure, form, and details generate or 'grow' by themselves – as Alan Dorin puts it:

'The generation of complex patterns from simple rules is a constant source of fascination for practitioners in the fields of computer generated imagery and sound. This interest takes root in a deeper drive to comprehend the world around us. Where we may, we reduce complex outcomes to the combination of simple principles. We look for causal connections between otherwise mysterious and unpredictable events. We seek order in chaos and, in our own work, seek *complexity for free*.' (Dorin 2000, 38).

## Markov Models:

Here I will introduce some important methodological approaches within generative or algorithmic art and design. By doing that I will concentrate on models that I use in the scope of this thesis and its practical examples (see chapter 2), namely on Markov Models and Generative grammars<sup>49</sup>.

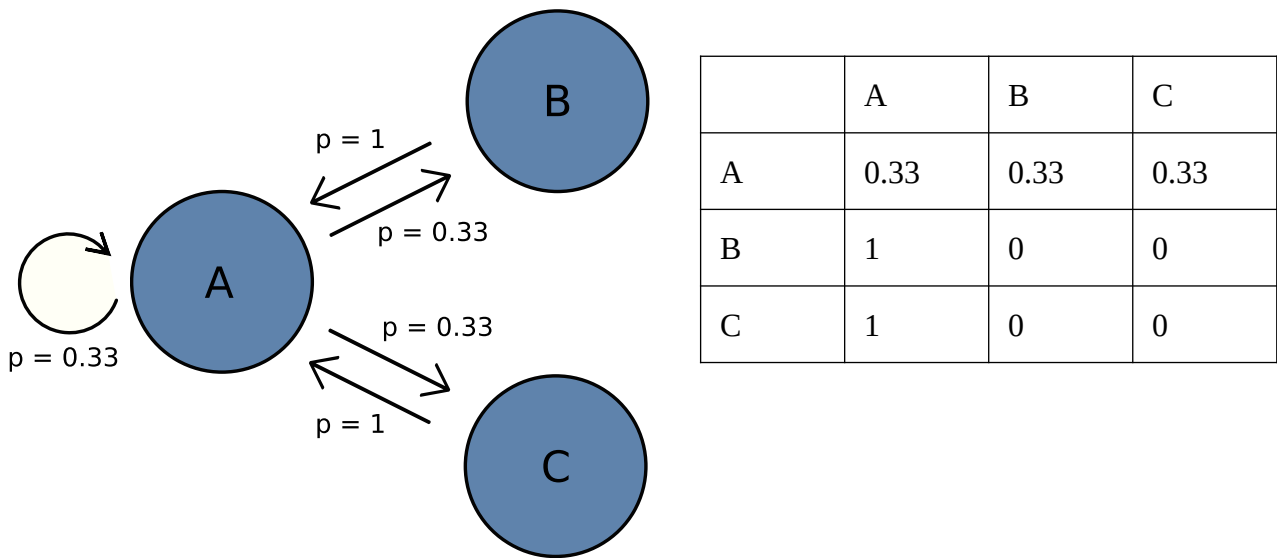
This is a stochastic model to describe the succession of elements of a system in terms of probability, invented by the Russian mathematician Andrey Andreyevich Markov (1856-1922). Markov Models are an interesting tool to analyze, describe and generate sequences of elements or states. Depending on the area of application these elements could be symbols, sounds, words, events, note values, etc. The elements or states are related by their probability of following one another. A sequence of these elements is also called a 'Markov chain', which is a stochastic one-dimensional sequence of states, where the future state depends on the current state. Every state in the system defines its possible next states by indicating the probability of the respective transition.

In other words, a Markov Model describes a set of states through a relation of succession of these states simply by indicating the probability of every possible succession. Given a limited sequence of states, the probabilities can be determined by analyzing the sequence. Take the sequence A --> B --> A --> A --> C --> A: Here A is followed by B, A, and C with a probability of 0.33, while C and B have the successor A with a probability of 1. The Markov Model that follows is shown in figure 13 with its transition matrix – a matrix showing all the possible successions in a set of states.

---

48 'Both in music and in visual art, the use of the term [generative art] has now converged on work that has been produced by the activation of a set of rules and where the artist lets a computer system take over at least some of the decision-making (although, of course, the artist determines the rules).' (Boden/Edmonds 2009, 4).

49 For a general overview of generative methods and models in algorithmic composition see (Nierhaus 2008).



**Fig. 13** Simple Markov Model with three states. Its transition graph on the left and its transition matrix to the right.

A Markov Model can be used for two things: To analyze an existing sequence to build a model (i.e. a transition matrix) and to generate a sequence out of a model (i.e. a transition matrix). In order to generate a sequence from a Markov model, one begins at the starting state and determines the next state according to the transition probabilities given in the transition matrix. This way a sequence of the desired length can be created step by step.

In the same manner, a text consisting of a sequence of words or a piece of music consisting of note values can be analyzed to create its transition matrix. On this base, a sequence of states (i.e. words or note values) can be generated to produce a sequence that is a kind of style imitation of the original sequence.

A Markov Model has a specific 'order' that determines how many past states are taken into account for determining the probabilities of the possible future states. If more than one past state is used to determine the transition probabilities, it is called a higher-order Markov process. The order indicates the number of past states that are included in the calculation. That means that higher-order models will reproduce an analyzed sequence more accurately at the cost of generative freedom. So, if the order is high enough the generation process will produce an exact copy of the analyzed sequence.

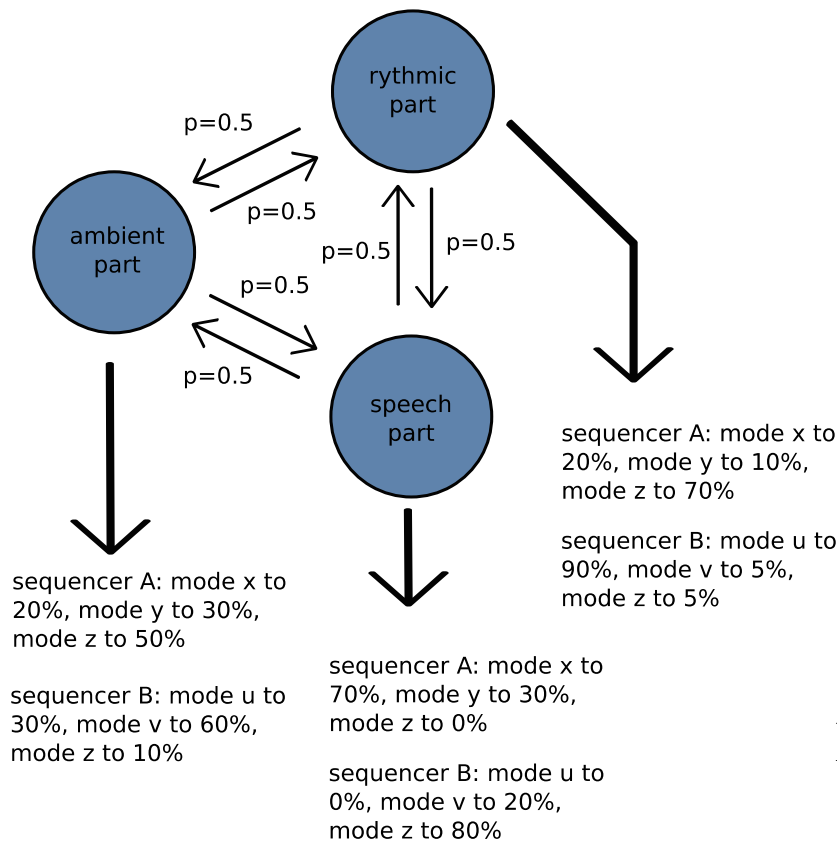
Another interesting concept is the 'Hidden Markov Model' (HMM). Here the states of the model are not equal to the output sequence, but the states in the model cause the output sequence through another (stochastic) process. So, the states of the model and its transitions are not visible, but they determine the visible output sequence through a probabilistic mapping. Generating with an HMM means creating a sequence of states within a Markov Model and then, in a second step, determining an output (event, symbol, value, etc.) through a stochastic process. For example, one could think of using an HMM for generating a sequence of harmonies (or harmonic states), from which specific

note values follow<sup>50</sup>. Figure 14 shows a possible sequence of harmonies corresponding to the states generated by the Hidden Markov Model and a possible output in form of note values. In order to determine the note values from the harmonies, we need another set of probabilities or rules which is not shown here (a simple one is to take random note values from the scale specified by the harmonic state)

hidden harmonic states	A <sup>maj</sup>	C <sup>min</sup>	D <sup>7</sup>	A <sup>maj</sup>
possible note outputs	a c# d	d# e# f	d c d	e c# a

**Fig. 14** Example for a sequence of hidden states and its visible output

Figure 15 shows an example of how an HMM could be used for structuring a musical piece or sound installation in functional, dynamic, or thematic parts. In this setup, the states of the Markov Model represent three functional parts that can be activated alternately. The states are 'hidden', while the 'modes' of the sequencers are 'visible', forming the musical surface of the sound piece. The changes between the hidden Markov states determine the modes played by the sequencers by setting different probabilities for the actually realized play-modes. The model in figure 15 does not show how the timing is organized as this is not important for the basic concept of an HMM.



**Fig. 15** Example of a Hidden Markov Model for switching between three functional states that control three sequencers.

50 For a more elaborate example for the application of HMMs in the generation of harmonic sequences by Moray Allen see (Nierhaus 2008, 78).

## Generative grammars

The concept of generative grammars is based on structuralist linguistics conceived in the early 20th century by theorists like Ferdinand de Saussure in Europe and Leonard Bloomfield in the US. Based on these approaches Noam Chomsky developed a linguistic model in the 1950s, which he called 'standard theory'. Chomsky's theory was based on a generative grammar model which had a big influence on linguistics, informatics, automata theory, and the theory of computation. He identifies four types of generative grammars and classifies them within the 'Chomsky Hierarchy' according to their complexity and generative capacity.<sup>51</sup> The productive process that is described by the formalism of generative grammars is based on a rewriting operation that is capable of generating a sentence (a sequence of symbols) out of grammatical rules. So, the idea of 'rewriting' in generative grammars describes the stepwise formulation of a phrase or sequence out of the rules that are defined in the particular grammar. Considering natural languages, we would therefore start from general elements like sentences, noun phrases, verbs, adjectives, and so on. These syntactical placeholders are called 'non-terminal symbols' and will be replaced by 'terminal symbols' or words in the last rewriting step. Forming or analyzing a sentence in this way unfolds a tree-like syntactic structure that is capable of determining if a certain phrase is part of the respective language (i.e. if it is correct according to the grammatical rules, which is called 'parsing' in informatics). A grammar tree also illustrates how a phrase can be produced out of simple rules by a rewriting technique. Such a stepwise replacement of symbols is appropriate to be formulated algorithmically. That is why it is an interesting model when identifying mechanisms for generative sequencing.

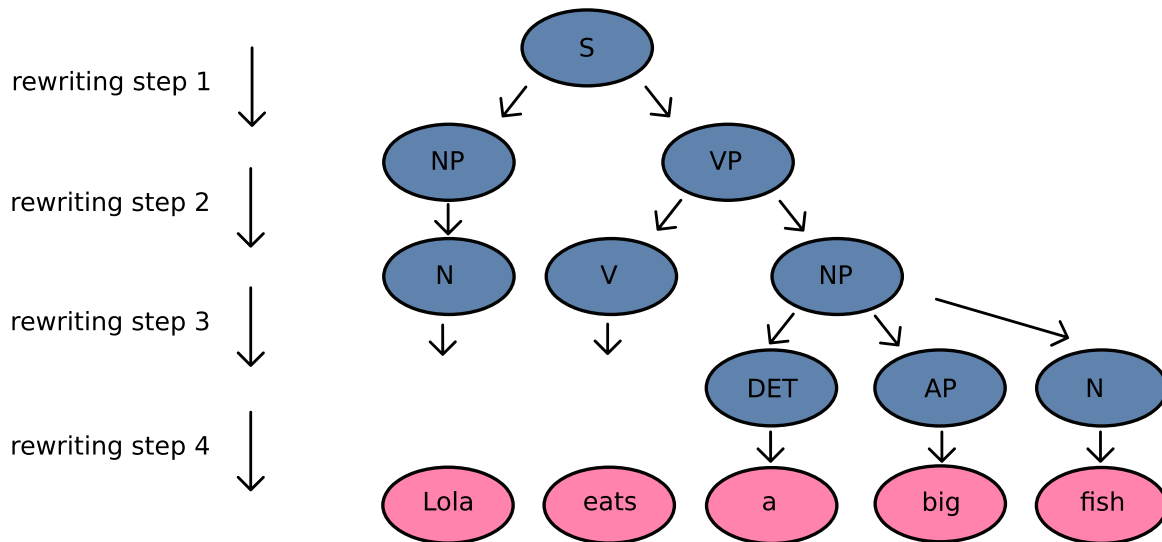
In order to clarify the generation formalism, I will give an example of the formation of a sentence. For the English language we assume the following rewriting or production rules (where S stands for sentence, NP for nominal phrase, VP for verbal phrase, PP for prepositional phrase, AP for Adjective Phrase, P for preposition, N for noun, V for verb, DET for determiner or article, Adv for Adverb, A for Adjective):

S	-->	NP	VP		
VP	-->	V	(NP)	(PP)	
AP	-->	(Adv)	A	(PP)	
PP	-->	P	NP		
NP	-->	(DET)	(AP)	N	(PP)

---

51 For a detailed description of the Chomsky Hierarchy and generative grammar in algorithmic composition see (Nierhaus 2008, 83f).

Figure 16 shows an example of how this process could be done for a sentence in English in form of a tree diagram: Starting with S the first rewriting step divides it into NP and VP. In the second rewriting step NP is replaced by N, VP is replaced by V and another N – and so on as defined by the production rules. In the last step, the non-terminal symbols are replaced by terminal symbols i.e. words, expressing a possible correct sentence in its final form, in this example: 'Lola eats a big fish'.



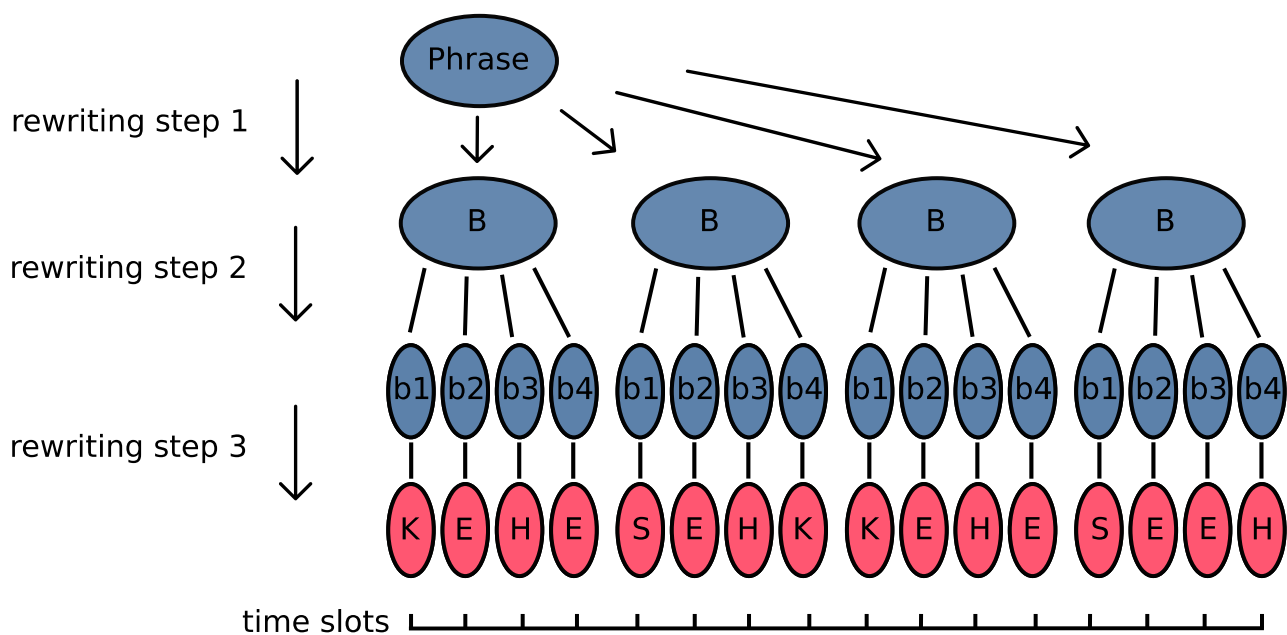
**Fig. 16** Example for a generative grammar in a natural language.

**Generating a rhythmic phrase:**

In my research and experimentation in sound design, I used the approach of generative grammars in some cases as a model for generating melodic and rhythmic structures or phrases. The generated phrases would preferably fit into a time grid with the length of a phrase restricted by the available slots in a certain time span. Figure 17 shows an example of a rhythmic phrase created with a non-deterministic generative grammar considering a loop with 16 points in time where it is possible to assign sound events. The grammar is non-deterministic because the rewriting rules contain various possibilities for replacing the non-terminal symbols. The terminal symbols of this grammar refer to sound events. In this example, I will use K for kick drum, S for snare drum, and H for a hat cymbal, E for an empty time slot.

The following rewriting rules contain the non-terminal symbols B for beat, and b1, b2, b3, b4 for the quavers within a beat:

- Phrase --> B B B B
- B --> b1 b2 b3 b4
- b1 --> K or S or E
- b2 --> H or E
- b3 --> H or S or E
- b4 --> H or K or E



**Fig. 17** Example for a rhythmic sequence generated by a generative grammar.

The use of generative grammar as in this example allows implementing hierarchical structures in the sense of grouping and relating elements of a sequence. While Markov Models decide about transitions from one state to another (or several states in the case of a higher-order MM) only in a 'horizontal' way, a 'lower order' generative grammar<sup>52</sup> allows defining hierarchical rules (like limiting a generation to a multiple of elements, defining groups of output elements depending on their position in the phrase, repeating sequences in variable length, etc.).

In chapter 2.5 I will give an application and implementation example for the use of generative grammar similar to the example above. Additionally, I will use probabilities in defining the rewriting rules to create a probabilistic grammar<sup>53</sup>.

Employing generative grammars to analyze musical material is a widely used method in musical analysis.<sup>54</sup> In generative sequencing the use of grammars is a way to formalize rules for pattern generation. In practice, however, I prefer thinking in more descriptive terms to formulate rules which define phrases through a production mechanism in a sequencer – for example:

- A phrase consists of 16 elements corresponding to 16 time slots.

52 The Chomsky Hierarchy of grammars divides grammars into four different orders according to their complexity. The simplest grammars (which are of the same complexity as Markov chains) are called 'regular grammars' or type 3 grammars. Lower order grammars (type 2, type 1, type 0) describe languages of higher complexity with fewer limitations regarding the formulation of production rules. For an overview of the Chomsky Hierarchy see (Nierhaus 2008, 87).

53 A probabilistic grammar is interesting for generating sequences because it allows to weight the various possibilities present in a rewriting rule to control the probability of generating different forms of strings. Probabilistic context-free grammars extend context-free grammars similar as hidden Markov Models extend regular grammars. For an example see (Bod 2001, 4).

54 See (Nierhaus 2008, 91f).



- The probability of an event on time slots 1,5,9,13 is 1, on 3,7,11,15 it is 0.5.
- All other time slots stay empty.
- Choose between events A , B and C, with probabilities of 0.5, 0.25 and 0.25.

Formulating these descriptive rules as generative grammar by defining them as rewriting rules is a possible way to conceptualize and represent the production formalism, but it has not necessarily to do with the practical design of a mechanism that generates the desired phrases algorithmically.

When considering the generative methods discussed here, Markov models and generative grammar, it is important to note that both methods originate in linguistics, and were designed to analyze or generate symbol chains. Such one-dimensional strings can be mapped to a sequencer in terms of timing and sound qualities like note values or the selection of specific samples, etc. When trying to describe or generate interaction between various sequencers present in a generative system, these linguistic methods are more limited.<sup>55</sup>

---

55 'Markov models, originating from linguistics, as well as generative grammars are in principle very well fitted for the processing of one-dimensional context-based sequences of symbols. On the downside, they are not very well fitted to account for dependencies between horizontal and vertical musical features (or, in general, across multiple dimensions).' (Nierhaus 2021, 15).

## 1.4 Sequencing in Puredata (Pd)

This chapter is intended to give a brief overview and insight for those who are not familiar with Pd or any graphical programming language<sup>56</sup>.

### Introduction to Puredata

Puredata or Pd is a visual programming language that was originally developed by Miller Puckette at the IRCAM in the 1990s. It was designed to enable the creation of interactive computer music and multimedia works. Miller Puckette is the main author of the program, but Pd is an open-source project with a large base of developers who work on new extensions. There are free versions for GNU/Linux, Mac OS X, iOS, Android, and Windows available. Pd can be used to process and generate sound, video, 2D/3D graphics, and interface sensors, input devices, and MIDI.<sup>57</sup> The main characteristics of Pd are the following:

**Pd is visual:** 'Pd is a visual programming environment, which means while using it you don't write code as such, but instead manipulate visual objects on the screen, connecting them into a system that produces some desired effect.' (Hillerson 2014, 5)

'Pd enables musicians, visual artists, performers, researchers, and developers to create software graphically without writing lines of code. Pd can be used to process and generate sound, video, 2D/3D graphics, and interface sensors, input devices, and MIDI. Pd can easily work over local and remote networks to integrate wearable technology, motor systems, lighting rigs, and other equipment. It is suitable for learning basic multimedia processing and visual programming methods as well as for realizing complex systems for large-scale projects.' (<https://puredata.info> – Jan 2021)

**Pd is a data flow programming language:** Control data and audio signals flow through Pd objects connected to each other and represented visually on the screen. As with most digital signal processing (DSP) programs, there are two primary speeds at which data is transferred:

- the audio rate, which typically runs at 44,100 samples per second
- the control rate, which operates at 1 block per 64 samples by default (so it would run at 1,5 ms by default).

**A Pd file or program is called a patch:** 'A Pd patch [...] consists of a collection of boxes connected in a network called patch.' (Puckette 2007, 15) So when opening a Pd patch one can see a data flow network consisting of wired boxes. Writing a Pd program or patch mainly consists of creating these boxes and linking them.

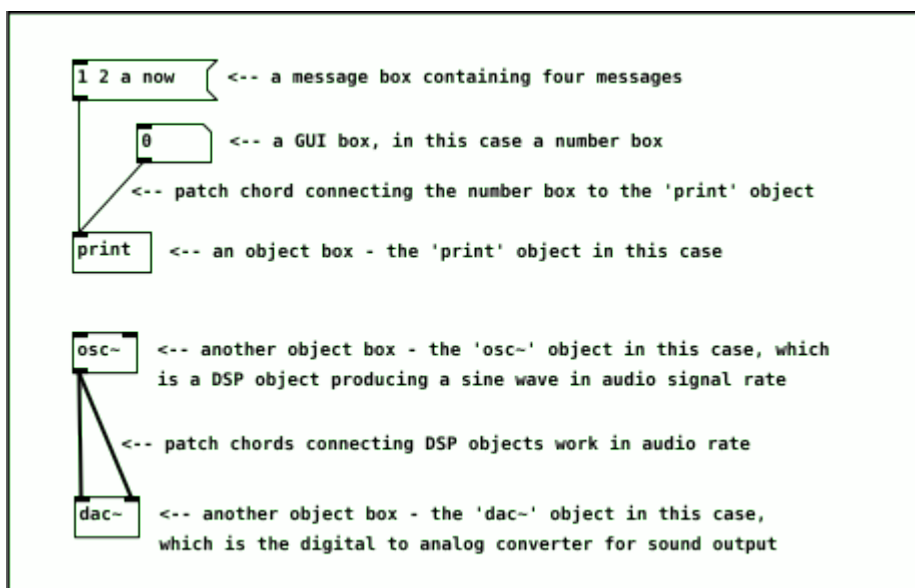
---

56 For extensive information see (Hillerson 2006), (Farnell, 2010), (Puckette 2007).

57 For a more detailed introduction to Pd by its creator Miller Puckette see (Puckette 2007, 15f). See also the webpage <https://puredata.info> (Jan 2021).

## Pd uses four types of functional boxes:

- Message boxes have a flag-shaped border, and they hold a message, which is sent through the patch cord whenever the message box is triggered. Messages supply commands and instructions to objects. The 'bang' message is a special type of message which plays an important role in Pd. It has no content but is necessary to trigger or initiate events. Often it is used as a start button to push the data into the flow.
- Object boxes have a rectangular border and contain Pd objects. The Pd objects are the algorithmic functions used to generate audio signals, operate on data streams, etc. Usually, they have an input and output slot and you can give them arguments to specify variables or functions in use. Pd comes with a variety of objects, ranging from mathematical and logical operators to general and special DSP functions.<sup>58</sup> If an object is designed to work on an audio signal, this is indicated by a tilde symbol (~) and the data stream it uses has to be in audio rate.
- GUI boxes are designed for user intervention and visual representation of data. This can be number boxes, buttons, toggles, sliders, etc. They are used for visually interacting with the patch by mouse clicking, dragging, and entering numbers or symbols.
- Comment boxes are used to write comments in the patch, they appear as the text you put in them and have no other functionality.



**Fig. 18** Pd example, different types of boxes in a patch.

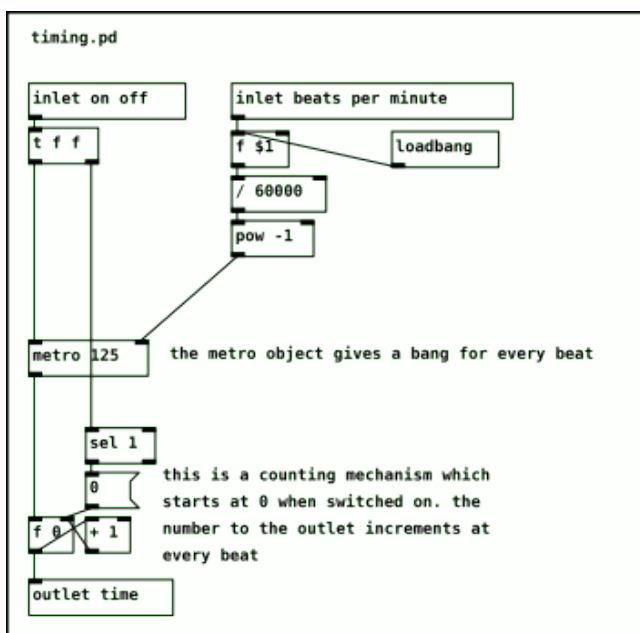
<sup>58</sup> 'Each object performs a specific task, which can vary in complexity from very low-level mathematical operations to complicated audio or video functions such as reverberation, FFT transformations, or video decoding. Objects include core Pd vanilla objects, external objects or externals (Pd objects compiled from C or C++), and abstractions (Pd patches loaded as objects)' (<https://puredata.info> – Jan 2021).

## Sequencing events in Puredata

In this section, the basics of sequencer design in Pd will be discussed<sup>59</sup>. Of course, there are countless possibilities to create patches for sequencing sound events. But mostly the function of a sequencer can be divided into two main areas: The timing and the generation of the utilized sounds.

### The Timing:

In standard sequencers, the timing functions in such a way that a timespan or loop is divided into smaller units. So, within the basic loop, several points in time are defined, which can be associated with sound events. Often, the sub-units of the loop are called beats.



*Fig. 19 In Pd a timing mechanism could be designed as shown in this example<sup>60</sup>.*

In the simplest case, such a setup in Pd works via a metronome that marks the basic beat. At every beat, the metronome gives an execution command – which is called 'bang' in Pd. The length of a beat can be set in milliseconds or bpm (beats per minute). In order to define a loop, it is necessary to determine how many beats it should contain. Mostly this measure is based on a 4/4 beat and 4, 8, or 16 ( $2^n$ ) time units are combined to a loop (which is therefore comparable to a bar in traditional notation). Each point in time in the loop must be clearly addressable and the loop keeps repeating itself. In Pd, this can be realized with a counter, which adds up the executed basic beats. The resulting increasing number is combined with a modulo function in the loop length. This way we get a repeating sequence of increasing numbers that indicates the position of the current beat in the loop.

59 Andy Farnell's Book 'Designing Sound' gives a comprehensive introduction to sequencing in Pd, see (Farnell 2010, 227f).

60 This example is a simplified version of an example from Andy Farnell, see (Farnell 2010, 228f).

Figure 20 shows a mechanism for summing up the desired number of beats to a loop. The resulting cycle of repeating numbers gives the current position in the loop (beat number), while the loop number indicates how many loops have already passed.

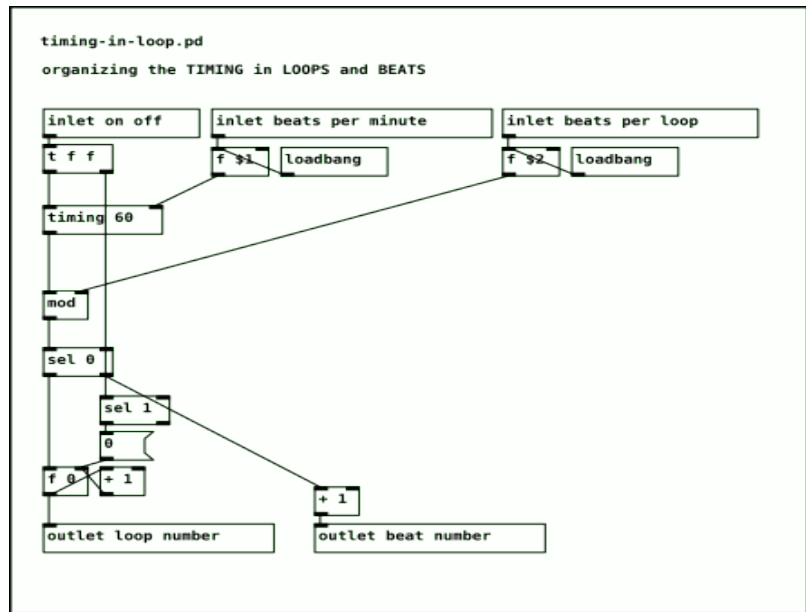


Fig. 20 A Pd patch for organizing beats in a loop.

The next step is an extension for marking every position by ON or OFF. In Pd, this can be done in the GUI, for example, by adding a toggle for each basic beat as shown in Figure 21.

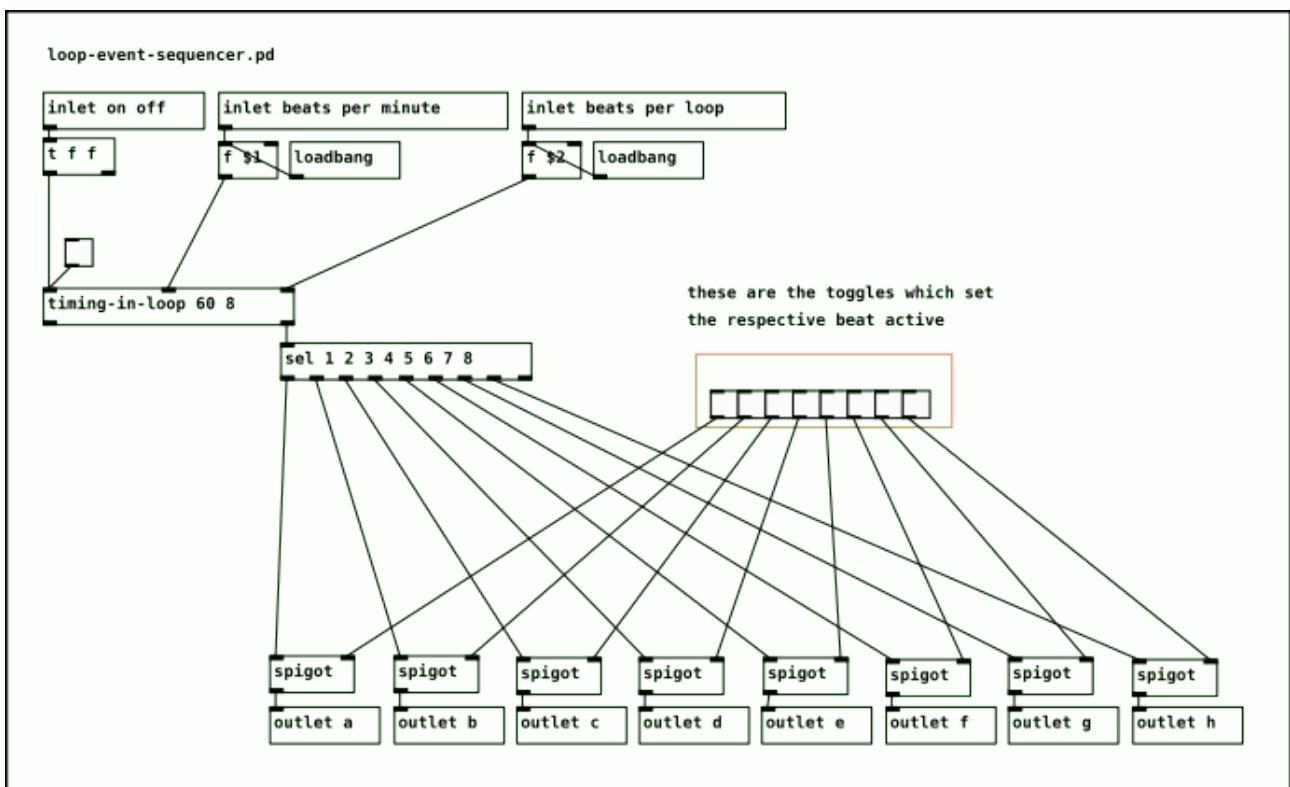
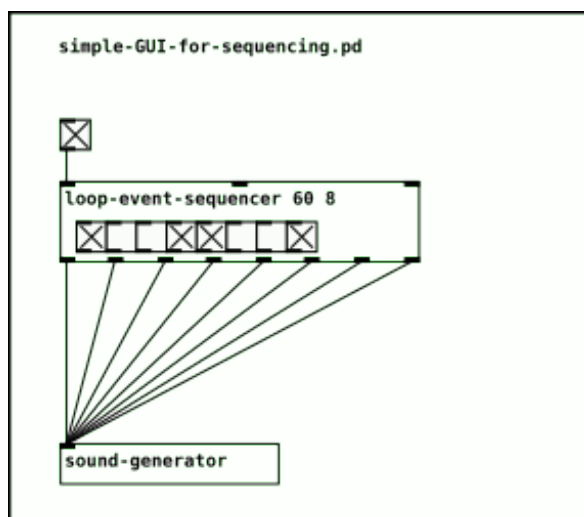


Fig. 21 Pd patch for marking eight beats as active or inactive.



**Fig. 22** The timing mechanism linked to a sound generator.

### The sound generation:

Apart from timing, the reproduction or generation of specific sounds is an essential aspect of designing a functioning sequencer. The generation of audio signals in Pd can work in many different ways. The two main methods are synthetic sound generation and the reproduction of existing audio samples with the help of a lookup mechanism. The synthetic generation of sounds can again be done in a number of ways, e.g. by applying an ADSR envelope to a continuous audio signal<sup>61</sup>.

Reproducing audio files works in Pd by importing sound files (in WAF or AIFF format) into an array<sup>62</sup>. These arrays are read according to the sample rate and reproduced as audio signals. A change in the reading speed causes a corresponding change in the pitch and length of the sample played and can be used in the sense of note values to create variations of a sound sample.

If a mechanism for generating the desired audio signal is now implemented, it can be linked to the timing mechanism. Thereby the audio signal or sound event can be triggered at any point in time defined by the timing mechanism as a beat.

61 See (Puckette 2007, 101f) and (Hillerson 2006, 41f) for more information on the creation and use of ADSR envelopes in Pd.

62 In Pd an array is a table with two values, x and y, where x is an increasing value from 1 to the array length. y is a value defined for all x. If an audio-file is displayed as an array, x is interpreted as a time axis according to the sample rate in samples and y as amplitude at the corresponding point in time.

## Chapter 2

# A generative sequencing system for algorithmic improvisation and its implementation in Puredata

### Overview

Chapter 2 of this thesis is dedicated to the presentation of a generative sequencing system that I developed when designing some generative sound installations in the past few years<sup>63</sup>. Writing this thesis gives me the opportunity to sketch a more general model that summarizes my various practical implementations from the past years and presents them more coherently. The generative setup I use allows both – live improvisation with generative sound sources (suitable for live performances) and the creation and editing of generative systems that work in a standalone mode (suitable for sound installations). The practice of producing sound with a generative system opens uncountable possibilities and can be adapted to specific situations and needs in a thousand ways. Here I will sketch a simplified model from my practical experience in order to get a better overview and more clarity on how such a system works and what its main components are. I will therefore explain the different elements of this model in their functionality and finally show possible implementations in Puredata.

After a short explanation of the concept of 'algorithmic improvisation' I will present a graphical model of the generative sequencing system, I am working with. In the following chapters I will explain this system in its essential components, which are:

- A **timing** strategy and the implementation of different timescales to enable the use of more complex sequencing strategies (chapter 2.1).
- A **communication medium** connecting the interface and the different components of the generative sequencing system (chapter 2.2).
- **Structure generators** using the time grid and/or user interventions for building and arranging structured sequences in the sense of musical streams (chapter 2.3).
- A **user interface** for two workflows and their combination: improvisation by the user, as well as the creation, display, and running of generative production systems (chapter 2.4).
- **Generative sound sources** including both, a mechanism for structuring symbolic sequences within the time grid, and a method to produce sound from these sequences (chapter 2.5).

---

<sup>63</sup> More information on the sound installations I designed applying the techniques of generating sequencing described in this thesis can be found on: [werkstatt.hotglue.me/sound](http://werkstatt.hotglue.me/sound) (Jan 2021).

## 2.0.1 Algorithmic improvisation

Algorithmic improvisation can mean two things:

- A **machine improvisation**: in this case, an algorithmic production or generation mechanism behaves like an improvisation by using non-deterministic generative mechanisms, random variables, probabilistic or stochastic methods.
- A **human improvisation** with machine-generated processes or content: Algorithmic production or generation mechanisms are coordinated and arranged by a human improviser.

Regarding my practical work in sound design, I would agree that both of these approaches apply. On the one hand, I am interested in creating machine improvisation by operating non-deterministic generative mechanisms that are capable of producing variations of an underlying structuring concept or idea. On the other hand, I enjoy the possibility of acting as a human improviser by triggering, combining, and influencing the generative algorithms in play. This means to keep some amount of control over generative production processes or an arrangement of such processes. These processes can be coordinated to form a more complex unity reflecting the creative imagination of the improvising artist. Depending on the intentions of a work of art (from a completely automatized sound installation to a human-controlled live performance) the side of machine improvisation or the side of human improvisation with generative contents will be more in the focus of creative capacity and aesthetic perception.

Improvisation – in music or other forms of artistic expression – can be characterized by two essential properties:

- It is non-deterministic: An Improvisation is characterized by open constraints<sup>64</sup>, that work with possibilities, probabilities, and conditions rather than fix rules. In practice, that means, that at some point uncertainty and spontaneity have to be involved in the creative process. Variability is considered more important than exact determination or reproduction.
- It is process-oriented: Essential decisions about the content are made during the process of production itself. In contrast to a composition, where one tries to define as many aspects of a piece as possible in advance, improvisation lives from consciously keeping the creative process always open<sup>65</sup>. This means that the creative process and the resulting piece itself are one and the same, and are not divided into two different phases. Creation and performance are identified in improvisation<sup>66</sup>.

---

64 Gerhard Nierhaus: 'To simplify, one could say that in contrast to a rule, which rather formulates a strict if-then relationship, constraints establish a network of conditions in which musical structure can evolve in manifold ways' (Nierhaus 2021/2).

65 The relation of composition and improvisation is complex and the space between these two poles is the creative place, where many works of art evolve. In reality, it is often a combination of composition and improvisation, that leads to interesting results. This kind of mixture is called 'comprovisation' by some theorists, see (Dale 2008).

66 For the aesthetic and ontological implications of such a process-orientated approach appreciating variation more than reproduction see chapter 1.1 of this thesis and (Eco 2002, 378f).



But the enhancement and emphasis of variation and the procedural character of improvisation are not to be equated with the absence of structure or constraints. It means to employ constraints that are designed to keep things in movement rather than to achieve a standstill (like for example introducing the contradictory rule, that every rule can be broken). The question if a computer can adequately simulate this kind of human creativity by using non-deterministic or stochastic production systems (or other algorithmic methods) for the generation of content is at the core of discussions on machine creativity and artificial intelligence<sup>67</sup>.

In my practical work as an artist, however, I found that it is an interesting method to combine one's own creativity with the (eventually simulated) creativity of a machine that employs non-deterministic generative mechanisms. It follows that a central aspect of 'algorithmic improvisation' – at least regarding my creative work – is precisely the combination and interaction of human and machine improvisation and creativity in the production process<sup>68</sup>. Applying this premise to the design of a generative sequencing system consequently means to keep in mind the two sides of algorithmic improvisation:

- Designing and realizing machine improvisation through the creation and usage of non-deterministic generative production systems.
- Developing possibilities for process-oriented intervention and control by the user as a form of human improvisation.

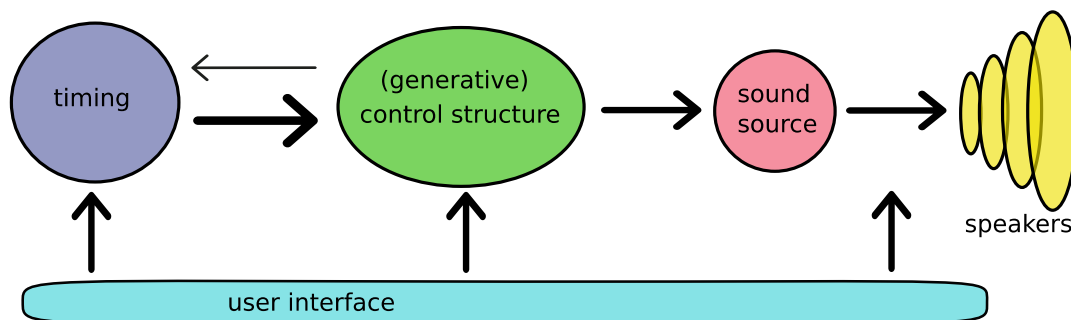
Combining these two aspects is the conceptual focus of the generative sequencing system described in chapter 2 of this thesis. This will be reflected in the system architecture emphasizing the importance of a graphical user interface (GUI) and a communication strategy that allows both, the human and the machine improviser to efficiently structure, arrange and combine generative sound production processes.

---

67 For a discussion on computational creativity see (Nierhaus 2021, 23f).

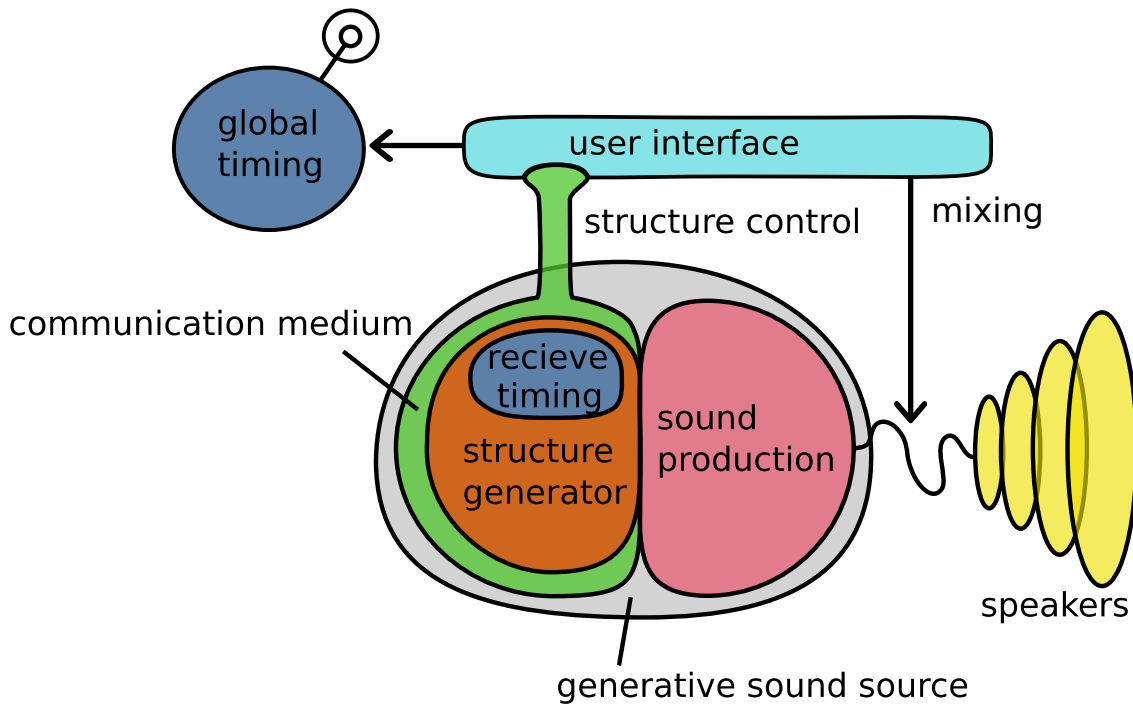
68 Nierhaus speaks about co-creation in this context. 'The generative approach is also often linked to an act of co-creation, wherein authorship of a composition can no longer be—or is intended to no longer be—attributed to the composer alone.' (Nierhaus 2021, 13).

## 2.0.2 A graphical model of a generative sequencing system for algorithmic improvisation



*Fig. 23 Simple model of a generative sequencer.*

Figure 23 shows a simple model of a generative sequencer. In the control structure, the timing is somehow transformed into structured sequences. The control structure operates on a sound source that can produce an audio signal that is amplified and output by speakers. It's easy to see in this model that the relevant things for generative production are happening in the control structure. The timing is just a modified clock or metro, and the sound source could be a synthesizer, sample play mechanism, etc. It seems obvious that for clarification and understanding of the system the control structure needs to be somehow differentiated and analyzed in its internal structure. Regarding my practical work with generative sound installations, I found that this is not an easy task. The processes I used for generating content are often specific implementations depending on the aesthetic goals of a work of art. Also, the possibilities and resources given by certain software and the choice of the sound material are decisive factors.



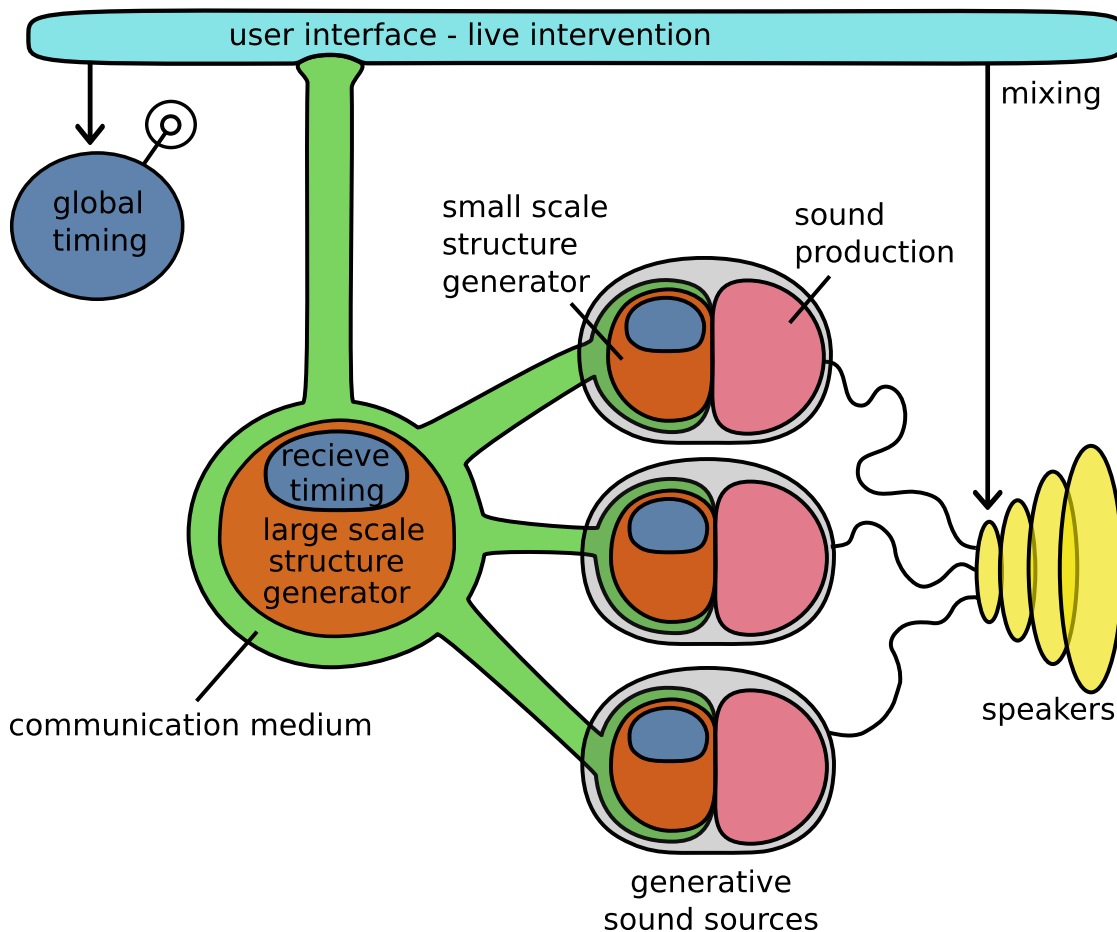
**Fig. 24** A generative sequencer built from a generative sound source connected to a user interface and a sound output.

Figure 24 shows a refined model of a generative sequencer. The control structure from the first model was differentiated into a structure generator and a communication medium. In the structure generator, the generation of sequences and patterns happens on a symbolic level. As we are always speaking about sequences in time, this structure generator has to be closely linked to the timing. Ideally, we would have a global timing controlled by the user interface that is also received by the structure generator. The communication medium is kind of a communication strategy connecting the different parts of the system.

Musical sequencers mostly are described as machines in which timing and structuring are already combined with the sound production mechanism (see chapter 1.2). So I found it appropriate to describe the union of a structure generator and a sound production mechanism as a 'generative sound source'. Also because the term 'sequencer' would be confusing when combining various sequencers to a sequencer system, which still would be a more complex sequencer. In the following, I will use the term 'generative sound source' in the sense of pattern generating sequencer connected to a sound production mechanism. The 'generative sound sources' can be imagined like single sequencers, virtual musical instruments, or other time-dependent sound production mechanisms. They have to be placed at the end of the generative system for mapping the symbolic information to an audible experience. As continuity in time is a decisive factor in the formation of musical streams (see chapter 1.1) it is suitable to treat these generative sound sources as essential components of the system. Only they can ensure continuity in audible experience by providing continuity in the mapping strategy. In other words, the generative sound sources are the base of the formation of musical streams.

Based on the model of a generative sequencer (figure 24) I developed a model for a generative sequencing system that contains various generative sound sources, as shown in figure 25. This model contains different control levels and timescales. A large-scale control operates on various generative sound sources present in a setup. According to my practical experiences, it is useful to differentiate the system control into at least two timescales: A large-scale control that concerns the longer parts of a sound piece or installation and the arrangement of various active sound sources. And a small-scale control where the generation of melodic or rhythmic patterns takes place. It is necessary to distinguish between these two timescales in order to obtain satisfactory results in terms of time stratification and complexity.

As illustrated in figure 25 in a generative system there would be at least one 'large-scale structure generator' that operates on several 'small-scale structure generators'. The communication medium serves as a kind of a nervous system connecting the large-scale structure generator with its subordinate generative sound sources. It also connects the user interface to all the structure generators present in the system. The timing for the whole system is controlled via the user interface and received by the structure generators on all levels of control.



**Fig. 25** Model for a generative sequencing system.

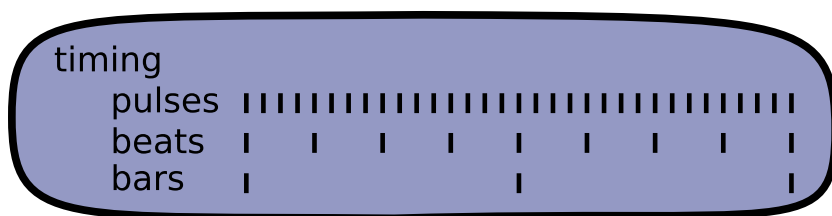
## 2.1 The timing: A basic pulse as a grid for variable loops

In sound and music design, timing is of course a very decisive variable – since sound is made out of waves with certain frequencies – which are defined by repetitions in time. When working with sequencers the timing is more about the triggering of sound events than the assembling of sound waves like in a wavetable synthesizer. But musical sequencers are usually also based on the repetition of a time period, often called a loop, divided into steps. These steps are used as triggers for sound events (see Chapter 1.3). The time periods in use are comparable in duration to notes or bars in traditional musical notation. However, timing in sequencers can also be referred to as frequency, as they use loops in time and positions within those loops to create rhythmic or melodic patterns. The widely used term 'beats per minute' is a common unit for frequency on a larger scale suitable for sequencing beats.

## 2.1.1 Pulses, beats, and bars

As pointed out in chapter 1.1 musical structure is essentially time-dependent and has to be organized on different scales for achieving structural depth in a piece.

The timing mechanism I propose for generative sequencing creates a time grid in which sound events can be placed and assembled to loops and variations from a minimal unit of time that I call the 'basic pulse'. All the sequencers involved in a setup would refer to this time-grid as a reference for their actions. To provide the system with different scales a number of pulses are summed to a beat and a number of beats are summed to a bar. Figure 26 illustrates the timing mechanism for creating beats and bars out of basic pulses. The shown example would correspond to a 4/4 time signature.



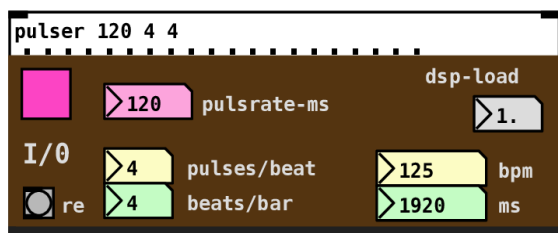
*Fig. 26 Time schedule in pulses, beats, and bars.*

The main advantages of the described timing strategy are the following:

- The basic pulse can be set arbitrarily – also to very small values when very short time intervals are desired. This is interesting for designing sounds that are not bounded to a traditional musical rhythmic structure – for example, tight groups or clusters of sounds within a short time interval.
- The number of basic pulses used to make one beat or one bar can be set as time grid for an entire piece or installation but every structure generating mechanism can be adjusted separately by setting the internal number of pulses to form a loop.
- The overall speed of a piece can be changed at any time for the whole system by changing the duration of the basic pulse.
- The flexible definition of 'beats' and 'bars' allows working with unusual time signatures. It allows the defining of longer temporal structures that always keep relying on the basic 'pulse' and its current length.

## 2.1.2 Timing implementation in Puredata

The Pd object that produces the pulses, beats, and bars that the generative system uses as a time grid is called the 'pulser'. The duration of the basic pulse and the duration of beats and bars can be changed at any time to change the speed of the sequences and processes that depend on. The use of such a central pulse generator has proven to be advantageous because it allows several temporally synchronous processes to be flexibly designed and manipulated without losing the coordination within a global temporal structure.



*Fig. 27* the 'pulser' object in Puredata.

In the 'pulser' object a 'pulserate' is specified in milliseconds in the pink 'pulserate-ms' number-box or given as the first argument of the 'pulser' object. When switched on, the 'pulser' starts to increment an integer number at every 'pulse'. This pulse number is perceived by all the structure generators involved in a setup and is used as the basis for temporal structuring. Additionally, a 'beat' and a 'bar' is defined. The 'beat' is composed by various 'pulses', the number of 'pulses' that form a 'beat' can be specified in the 'pulses/beat' number box or given as its second argument. The 'bar' is composed of various 'beats'. The number of 'beats' that form a 'bar' can be specified in the 'beats/bar' number box or given as its second argument. In the 'bpm' number box to the right, the resulting beats per minute are shown. The 'bpm' number can be changed and set in the GUI. This will adjust the 'pulserate' to achieve the desired beats per minute. To the lower right of the 'pulser' the resulting length of one 'bar' is shown in milliseconds. The 'beat' and the 'bar' were implemented to simplify setups that use multiples of a basic pulse for temporal structuring. In the GUI of the 'pulser' there is also a 'dsp-load' number-box that shows the mean CPU load.

In the example shown in figure 27, the pulse is 120 milliseconds long and four pulses will form a beat, which results in 125 beats per minute. A bar lasts 4 beats and is therefore 1920 milliseconds long.

## 2.2 The communication medium: Connecting user interface, structure generators, and generative sound sources

Based on the timing structure made from pulses, beats, and bars I began developing a communication strategy suited for the organization and connection of different timescales and control layers. Like this it should become possible to organize the sequencing system in a hierarchic tree-like structure: Short periods of time should be organized according to large-scale sequences and small structures should be grouped and arranged to form larger structures.

To achieve this goal I experimented with interconnected switches or finite-state machines that can be accessed both via the user interface as well as by an automated control structure (i.e. the structure generators on different scales in the system). These switches would finally control the play-modes of the involved generative sound sources either by being activated manually or automatically. A system should be able to include several generative sound sources, like the model in figure 22, and it should be possible to perform the following operations:

- Switching a generative sound source to a specific generative play-mode: So the switch had to include two variables: The identity of the sound source and the play-mode to switch to.
- Switching several sources at the same time via the GUI: That means that the switches have to have some marker to prepare various of them for switching at a desired point of time.
- Going back to the former state of all involved sound sources: So the switches need to have a memory or stack that allows stepping back to a former state.
- Connecting the different sound sources to form a network that could serve as a large-scale structure generator: To achieve this, the switches have to be connectable to other switches and to logical operators.
- Switching one or several structure generators: So the same switches have to be suitable for switching generative sound sources as well as for switching the large-scale structure generators.
- It should be possible to execute all the switching operations by hand via the GUI or by an automatic structure generator.

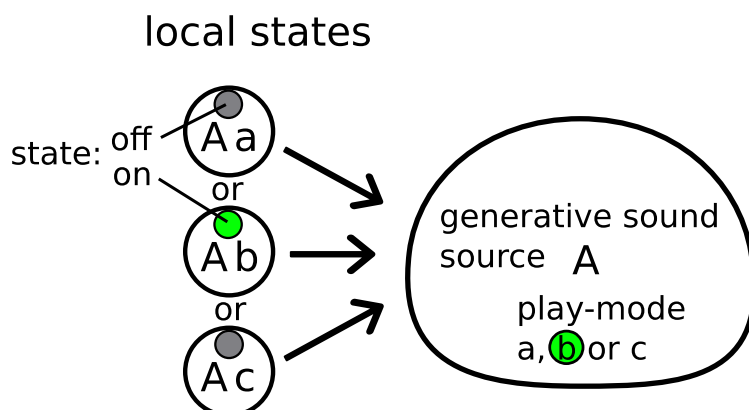


## 2.2.1 Communicating finite-state machines: The 'local states'

To meet these criteria I conceived special switches which I will call 'local states' in the following. The local states are the communication medium between the generative sound sources and a structure generator operating on them. They are also the communication medium that connects the user interface to a large-scale structure generator and the generative sound sources. Like this, the user becomes also a kind of structure generator, that uses the local states to control the sound sources. This allows the design of finite-state transition networks or tree-like structures that resemble generative grammars.

The idea of using 'local states' as I describe them here came up when I tried to organize the interaction of various generative sound sources for a sound installation. I used recordings of speech in superposition with field recordings that should be followed or replaced by different rhythmic patterns by certain probabilities and so on. So I had a lot of different pre-edited (more or less generative) 'play-modes' i.e. instructions for the generative sound sources of the system. In that situation, I experimented with a setup that should allow defining which play-modes could sound together and which are logically excluded because they belong to the same sound source (like playing two melodies at the same time on one 'instrument'). Working on this problem I ended up using 'local states' that are always associated with two variables: the sound source and the play-mode. When switching to a specific play-mode, any other active play-mode (if there is) of the same source must be switched off first so that there is no confusion. This is also useful when improvising with several sound sources and play-modes in a live situation. Naturally, a sound source can only realize one play-mode at a time, excluding all other possible play-modes. And all present sound sources can simultaneously realize one of their play-modes without any logical exclusion. This fact is reflected in the local states as shown in figure 28.

The local states are also reacting to a 'global state number' in terms of keeping or changing (refreshing or rewriting) their actual state when the global state number is increased. This function makes them much easier to organize in human live improvisation (see chapter 2.4).

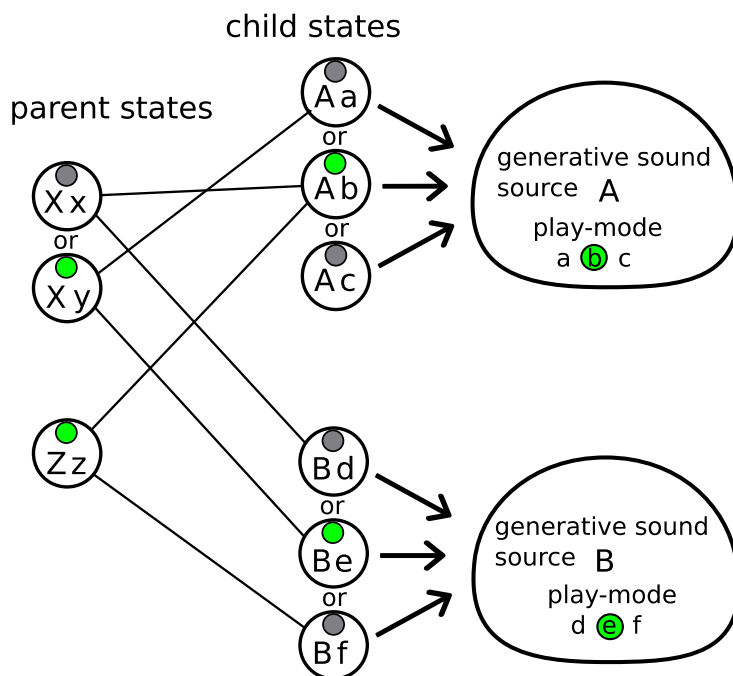


**Fig. 28** The local state specifies a target (sound source or structure generator) and an exclusive play-mode.

## 2.2.2 The grouping of play-modes: building systems, networks, and hierarchies

The grouping of local states means combining different local states into a superordinate or parent local state. When the parent state is switched, all connected child states are also involved. Such hierarchic grouping relations are an essential component of structuring – both, in terms of perception and production of structured content (see chapter 1.1).

The grouping to a parent state can be designed in different ways. It may contain probabilities, exclusive possibilities, or a certain chronological order within the child states. In this way, it is possible to build structure generators, that are based on local states in combination with logical operators and timing information (see chapter 2.3).

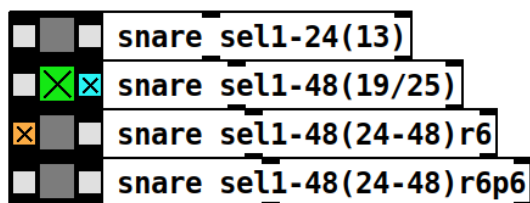


**Fig. 29** Parenting local states to form a simple control network.

Obviously, the switching of such a state transition network could be formulated as a generative grammar, where the parent states would be non-terminal symbols and the child states would be terminal symbols which are then mapped to a perceptible form by the generative sound sources. The result of the network would be a kind of sentence, a sequence of the active play-modes at a certain point in time. In this view, however, the sequence produced by the grammar or network would not be a temporal sequence, but a simultaneous arrangement. In a more complex system, there are mostly constraints concerning the temporal sequence and concerning the synchronous arrangement of simultaneous elements. On the question of the distinction between diachronic and synchronic structure, see chapter 2.3.

## 2.2.4 Implementation in Puredata: The 'losta' object

The local state object 'losta' is in principle a finite-state machine with an ON and an OFF state and a memory (or stack) of all its past states that allows it to switch back to the former state. The state of a 'losta' is shown and can be switched via its main toggle (green in figure 30). The special of the 'losta' object is, that it is identified by its arguments<sup>69</sup> and the way it is able to interact with other 'lostas' in a generative setup. So the arguments (in figure 30 'snare' is the first argument, and a shortcut referring to playing instructions is the second) are comparable to 'instrument' and 'play-mode' in a musical setup. For practical reasons the 'losta' is designed as one switch for every possible 'play-mode'. As the first argument refers to a generative sound source, various 'lostas' in a system will interact respecting a logical 'OR': only one can be in an ON position at a time.



*Fig. 30* A local state group of four 'losta' (as an abbreviation for 'local state') objects in Pd.

Furthermore, the 'losta' object can be used in various sub-patches in any number of copies. Any 'losta' having the two arguments identical is switched to the same state, respecting a logical 'AND'. This makes it possible to control multiple instances of 'losta' in and from different sub-patches<sup>70</sup>. Anyway, the main idea of using these interacting 'losta' objects is to be able to build various parallel or exclusive control structures (i.e. structure generators) that can be addressed manually or automatically.

In live improvisations with sequencers, it is certainly practical to switch between different pre-edited generative play-modes for keeping an adequate amount of control. Doing that, it is necessary to switch OFF other play-modes of the same sound source first. This function allows to easily define mutually exclusive play-modes that can not be executed at the same time.

<sup>69</sup> Pd objects have often one or several arguments specifying the function of the object. These arguments stand simply after the name of the object separated by a space, they can be numbers or symbol strings, see chapter 1.3.

<sup>70</sup> In my setups normally I use a main-patch – which is the user interface in the model of figure 25 where I have an overview of all play-modes and where I can control important parameters of timing and mixing. In this user interface, I would put an instance of every 'losta' in use to be able to activate all the play-modes used in an installation or sound piece by hand. Then I would use a sub-patch where the sound sources are connected with the corresponding 'lostas' (i.e. play-modes) setting the actual play-modes via messages. Finally, I would at least use one more sub-patch for building a structure generator for the large-scale timing and interaction of the involved 'lostas'. See chapter 2.5 for more details on the user interface.

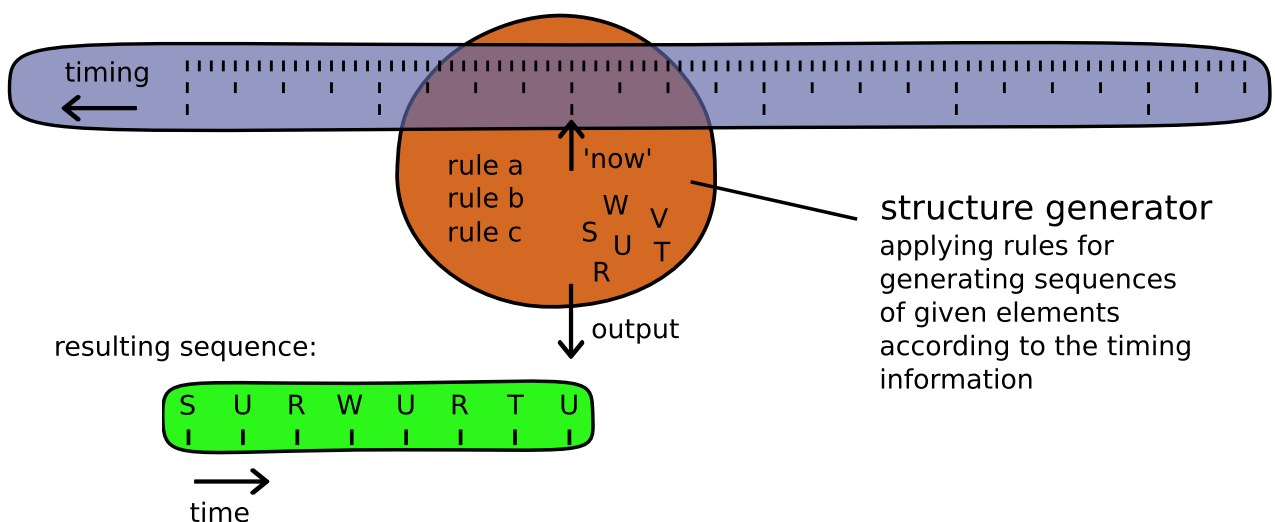
Another important feature of 'losta' is its recognition of the 'global state number' and its changes. The global state number is the reference number of the additional toggles in the GUI of the 'losta'. The leftmost (orange in figure 30) toggle specifies the state of the respective 'losta' at the past global state, while the rightmost toggle (light blue in figure 30) specifies the state at the next global state. As this function is more decisive for the creation of a user interface for human improvisation it will be discussed in chapter 2.4.

## 2.3 Structure generators

For the automated generation of structured sequences, it is necessary to design mechanisms that produce these structures on a symbolic level. These mechanisms – holding and applying the instructions required to produce the desired structures – I call 'structure generators' in this thesis<sup>71</sup>. Depending on the position in the generative system the structure generators work with the systems timing information and/or other trigger or step mechanisms accessible within the GUI. The structure generators can be designed in many different ways, using diverse generative approaches and methods.

It is possible to describe a structuring process by the use of generative rules or constraints. When starting from a set of possible events (like the words of a language) the most unstructured and unrestricted way to sequence these events would be a random function that does not establish any relations between the succession of the events. As specified in chapter 1.1 structuring a musical sequence means establishing horizontal and vertical relations to interpret (perceive and understand) and/or generate the respective sequence.

Horizontal relations are conditions or links between elements of the form: 'When A happens B happens at the same time or at the next step' (like in Markov chains or regular grammars, see chapter 1.2). These relations or rules can contain probabilistic selection out of various possibilities. A vertical relation is the formation of groups of events or elements which can be repeated, varied, or grouped again. A more complex generative system would also use conditioning constraints to make decisions depending on other changing parameters or activities in the system. As mentioned in the last chapter, a setup with interconnected local states is a very adaptable scenario that allows using different mechanisms, algorithms, and methods to organize and interconnect generative sound sources and their play-modes.



**Fig. 31** Model of a structure generator linked to the system timing.

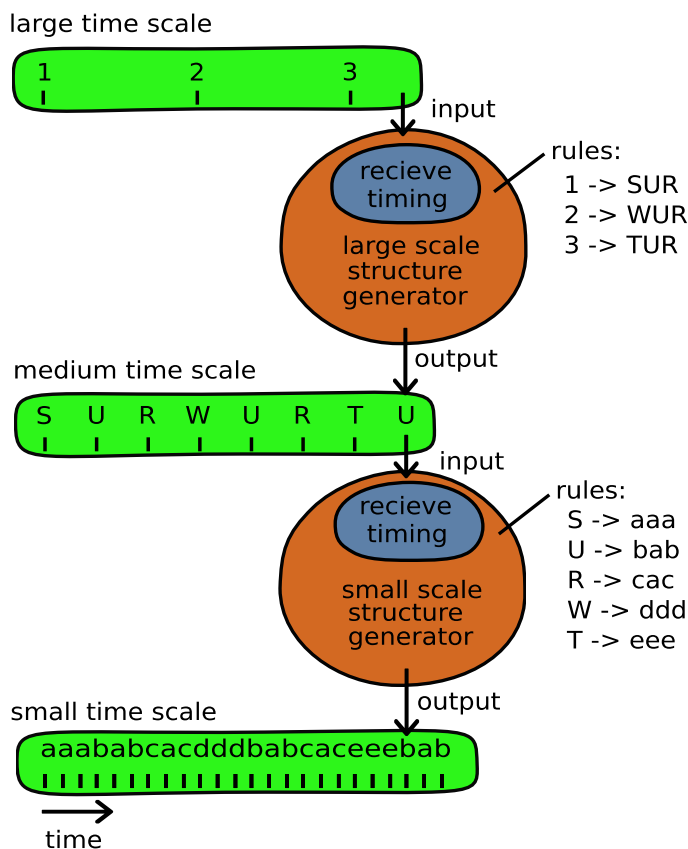
<sup>71</sup> Following Karlheinz Essl, who used this term in his 1996 work of the same name in German: 'Strukturgeneratoren' (Essl 1996). See chapter 1.1.

Figure 31 shows a graphic model of how a structure generator interacts with the system timing to generate a sequence from available elements. Depending on the architecture of the structure generator (which is not specified in this model), the rules could contain information about the placement within the timeline, the succession of elements, repetition and variation.

### 2.3.1 Small-scale and large-scale structure

For being a structure generator (and not only a pattern generator) the possible definition of vertical relations like the grouping of elements is essential. In figure 32 the output elements of a structure generator (R, S, T, U, V, W) are used as input of a further structure generator on a smaller scale. The result is a tree structure, where 'larger-scale' structures contain or manipulate 'smaller-scale' structures. I use 'smaller' and 'larger' here to emphasize the capacity of relating different scales of timing or grouping. As shown in figure 29 the local states provide a communication medium for this purpose by enabling grouping and parenting.

In the generative system described here, the 'smallest-scale' structure generators are situated within the generative sound sources. They produce rhythmic and melodic patterns specified by their play-mode. The 'larger-scale' structure generators are organizing sequences, overlays, and interactions of play-modes of generative sound sources via the local states.



*Fig. 32 Model of structure generators interacting on different timescales.*

## 2.3.2 Synchronous and diachronous structure

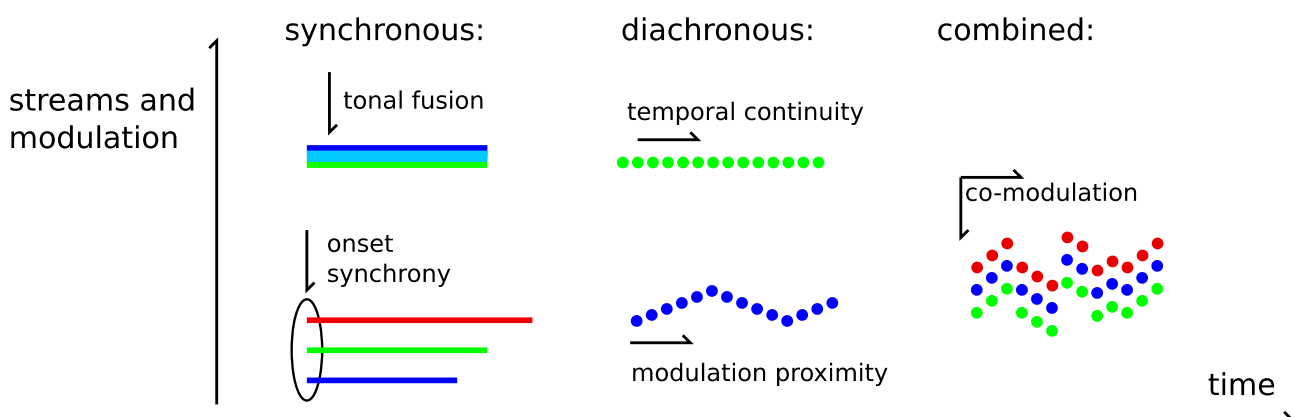
An important aspect when speaking about musical structure is the differentiation of the synchronous and the diachronous structure and its interaction<sup>72</sup>. As pointed out in chapter 1.1 the perception and production of structure within a highly time-dependent medium such as sound is to be imagined more as an organized flow of interconnected streams than a static system. The creation of time-dependent structures or streams must always take two factors into account:

- The factor of succession/evolution or **diachronicity**: How to create a stream and its movements.
- The factor of simultaneity or **synchronicity**: How to coordinate several parallel events and streams in terms of interaction and overlay. How to create a group (or stream) of streams.

The formation of musical streams relies on various factors or principles<sup>73</sup>. These principles are either synchronous, diachronous, or both (figure 33):

Synchronous principles	Diachronous principles	Combined principles
The principle of tonal fusion	The principle of temporal continuity	The (pitch) co-modulation principle
The onset synchrony principle	The (pitch) proximity principle	

**Fig. 33** Table of synchronous and diachronous stream formation principles.

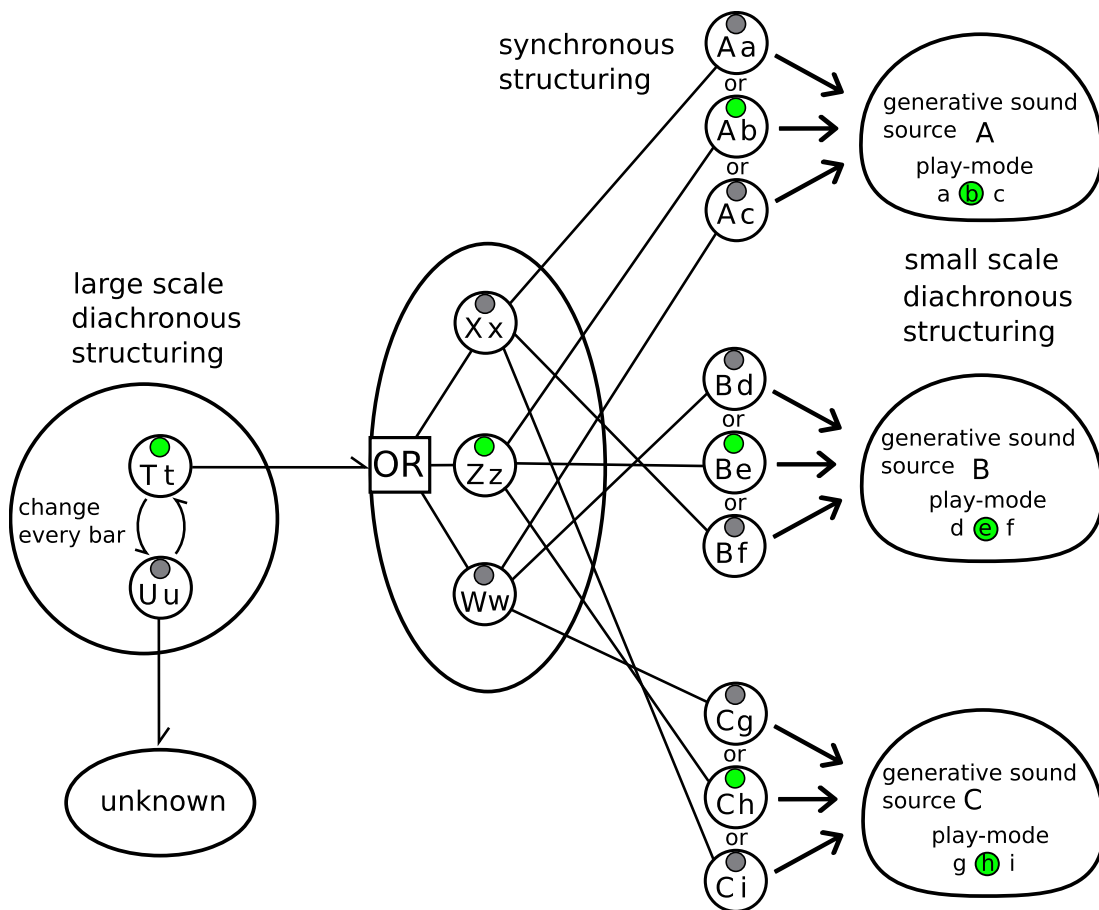


**Fig. 34** Schematic model of stream formation factors along a horizontal time axis.

72 Cambouropoulos analyzes the interplay of diachronic and synchronic grouping in stream formation and states: 'In a sense, there is a competition between the vertical [i.e. synchronous] and horizontal [i.e. diachronous] principles of auditory grouping. It is exactly this competition that makes it difficult to describe processes of auditory streaming systematically.' (Cambouropoulos 2008, 78).

73 In this thesis I describe the most important factors for perceiving, separating, and creating musical streams according to (Cambouropoulos 2008), see chapter 1.1.

For the sequencing system described here, one can assume that the generative sound sources correspond roughly to musical streams, as they are designed for producing a sequence of closely related sounds. In this sense, the structure generators situated within the generative sound sources are based on diachronous stream formation factors<sup>74</sup>. When it comes to organizing the interaction and the large-scale behavior of several generative sound sources synchronous structuring becomes more important. Figure 35 shows a model where the synchronous and diachronous structuring is illustrated using the local states introduced in chapter 2.3. Here the large-scale diachronous structuring is an alternation of the local states 'Tt' and 'Uu'. 'Tt' is linked to a synchronous structuring mechanism, that activates one of three possible local states (Xx, Zz, Ww). Each of these local states is linked to a group of play-modes. Each of these local states is linked to a group of play-modes.

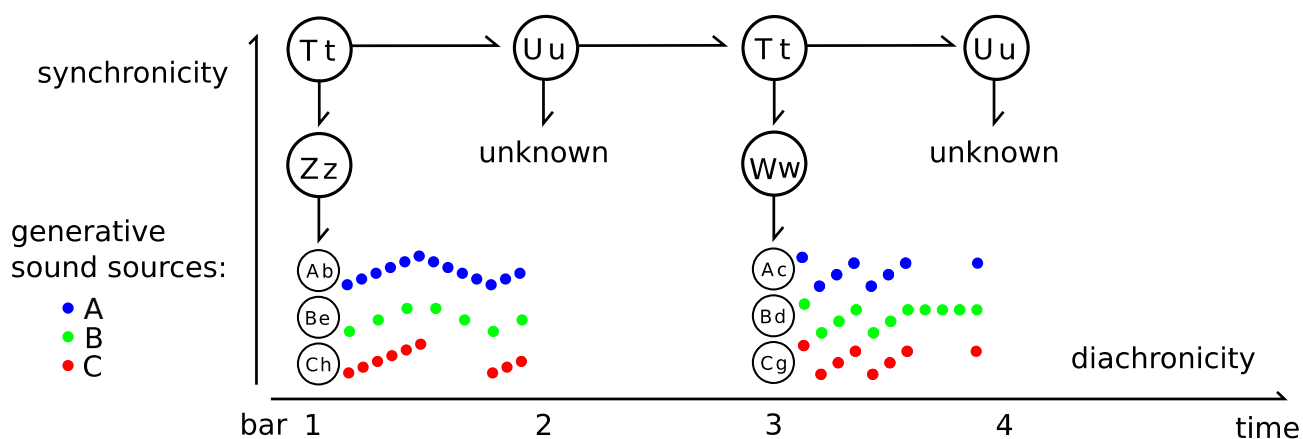


**Fig. 35** Diachronous and Synchronous structuring using local states.

<sup>74</sup> As pointed out before, the generative sound sources are similar to musical sequencers with generative extensions. Such musical sequencers are mostly designed to create one-dimensional musical streams by arranging sound-events within a time loop.



In this example, there are three generative sound sources that can play at the same time. The synchronous structuring concerns the decision about which play-modes can be executed together and which ones cannot. A symbolic representation of this process is shown in figure 36. Two alternating states produce a synchronous group (Ab/Be/Ch) of active states. The symbols or states of the synchronous sequence are finally transformed to various diachronous sound sequences or streams by the generative sound sources. The most obvious stream formation factor here is the onset synchrony principle, as the three parallel play-modes start at the same time. Further factors for stream formation or separation should be considered in the definition of the commonly realized play-modes, here represented by the distribution of colored dots.



**Fig. 36** Diachronous and synchronous stream formation according to local states and generative sound sources.

### 2.3.3 Implementation examples in Puredata

#### The 'Timed Structure Generator' (TSG): Structuring diachronous sequences

For the purpose of structuring sequences that can be used on different timescales (also for the large-scale control of a generative system), I implemented a Pd object called 'Timed Structure Generator' or TSG. This sequencer generates timed strings of numbers which is a further step that can be used for creating sequences and arrangements of sound sources and their play-modes. The output is numbers that can be redirected via the communication medium (the local states, see chapter 2.2) to form structure generating networks (see figure 35, 42, 44). The timing can refer to the bars, beats, or pulses given by the system timing (see chapter 2.1) and the output sequence will be timed within a loop. The length of the loop is set by the number of time slots it contains.

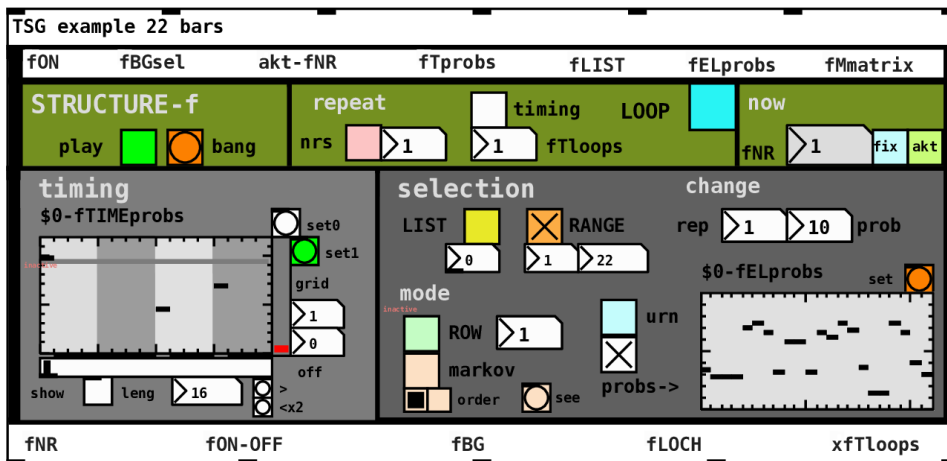


Fig. 37 The 'Timed Structure Generator' (TSG) in Puredata.

#### The 'timing' section:

In the timing section (fig 38) the events or outputs are set as probabilities. In the example shown, there are eight time slots and a black bar which enables to set the probability of an event at the corresponding time.

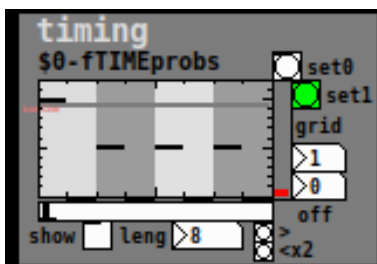
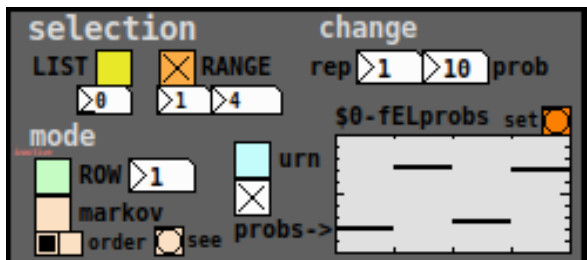


Fig. 38 The 'timing' section allows setting the output or event timing as probabilities.

### The 'selection' section:

Here the set of possible output numbers can be limited within a 'RANGE' or entered as a 'LIST' via the corresponding inlets or by message. The mode of selecting the output numbers can also be specified. The possible modes are:

- 'ROW': The output numbers will form a row. In the number box, the step size can be defined.
- 'markov': Finds the output numbers with the help of a transition matrix, which specifies the probabilities of a direct succession of the output numbers (as this is done in a Markov Model). The transition matrix can be entered via the rightmost inlet or by a message.
- 'urn': This refers to a random with memory (outputs all possible numbers randomly before repeating a number).
- 'probs': Here, the numbers will be selected as specified in the probability array to the right, where the probability for every possible number can be set.



*Fig. 39* The 'selection' section defines how the output numbers will be chosen from the pool of possible numbers.

### The 'repeat' section:

Another important structuring tool implemented in the TSG is the repeat function (in the upper center of the GUI). Here one can force the TSG to repeat one or several output elements regarding timing and/or selection. The repetition of one or several entire loops is also possible.

## Tools for arranging synchronous groups

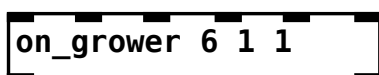
For this task, it is possible to use the local states and connect them via grouping mechanisms to various child states (see figure 35). In the following, I describe two grouping mechanisms that I implemented in Pd for the arrangement of synchronous groups of active states.

The '**urn\_on\_distributer**' object sets a number of active states within a set of possible states. The selection algorithm is random with memory ('urn'). The arguments of the object define the number of possible states and the number of states to be set to ON. The output is a sequence of pairs of numbers, which specify a state number and its state (1 or 0). This output can be connected to a 'route' object to form a group of possible local states (like in figure 42) whose states (On or OFF) will be set according to the 'urn\_on\_distributer' object. In the example shown in figure 40 five states are given, two will be set to ON and three to OFF when 'urn\_on\_distribute' is activated. The distribution is recalculated whenever the object receives a 'bang'. Like this, it is possible to map a timed sequence to a changing distribution of active states.

A screenshot of a Pd object named 'urn\_on\_distributer' with two arguments, '5' and '2'. The object is shown in a rectangular box with a black border and a white background. The text 'urn\_on\_distributer 5 2' is displayed in a monospaced font.

*Fig. 40* The 'urn\_on\_distributer' object in Pd.

The '**on\_grower**' object also performs a grouping operation that defines a number of active states within a set of possible states. It increments or decrements the number of active states whenever receiving a bang in the fourth inlet. The arguments of this object are the number of possible states, the number of active states when starting, and the number of states to set active or inactive in the next step.

A screenshot of a Pd object named 'on\_grower' with three arguments, '6', '1', and '1'. The object is shown in a rectangular box with a black border and a white background. The text 'on\_grower 6 1 1' is displayed in a monospaced font.

*Fig. 41* The 'on\_grower' object in Puredata.

These synchronous grouping objects can be connected to local states to build networks for organizing the play-modes of various generative sound sources and their overlays. In figure 42 a central 'on\_grower' object is used to switch six groups of local states. 'urn\_on\_distribute' objects are used to choose the active states within these groups. In this practical case, the groups correspond to six instruments of a drum-set or percussion group.

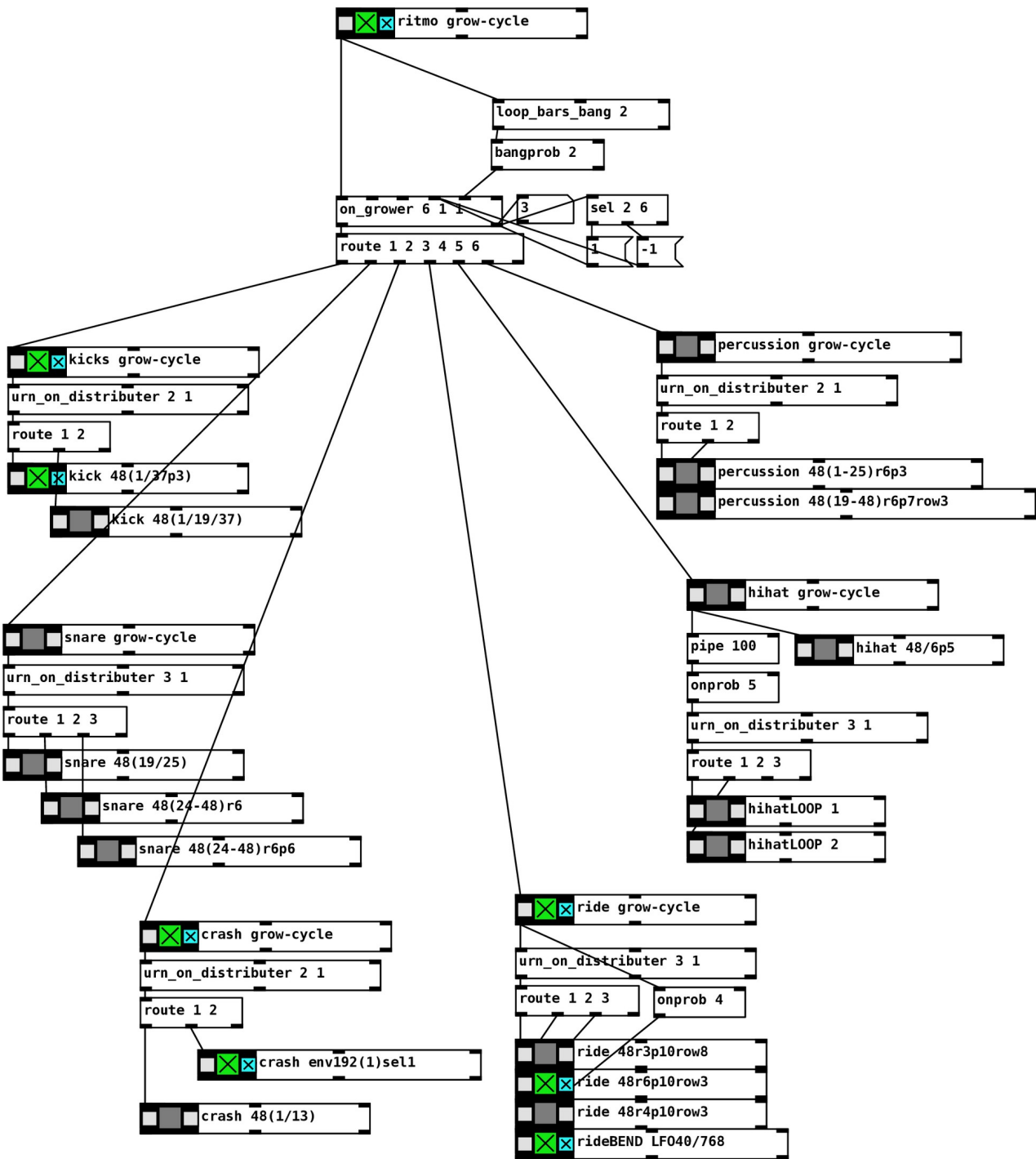


Fig. 42 A network of grouping objects connected to local states

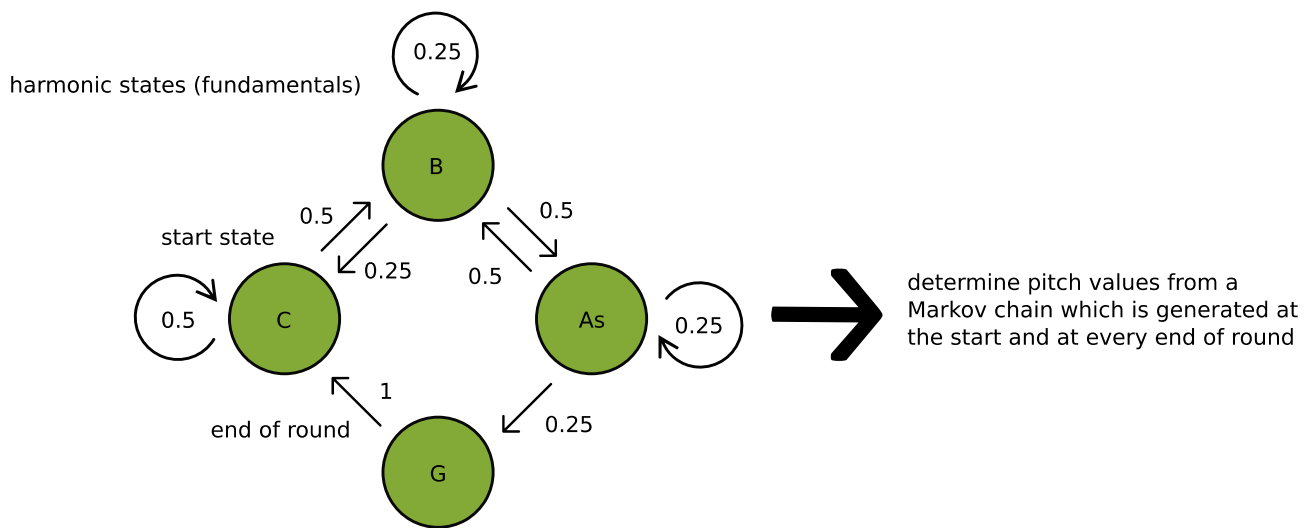
## Example of a large-scale structure generator based on a Hidden Markov Model

To give an example for the setup of a Hidden Markov Model (HMM) I will describe a patch, where I use a Markov-chain of four states that gives a harmonic background to the bass-line and the melody-line by setting different harmonic scales from which pitch values can be chosen via a stochastic process (see chapter 1.2 for a description of the HMM approach). So there will be a MM that generates the changes between different scales, in this case, C, B, As, and G. The succession scheme or basic movement model should be a modified circle (see figure 44), with a possible step in both directions. The 'clockwise' direction in our circular model will get a chance of 50 %, the 'counter-clockwise' direction 25 %, and the chance of staying in the same state will be 25 %. An exception to this should be that As should have only a chance 25 % chance to go to G and 50% of going back to B. G should always go to C directly, and C cannot go back to G (like marking an endpoint of the circular movement not happening so often as the other changes) Given this information the transition matrix of the MM looks like this:

	C	B	As	G
C	0.5	0.25		1
B	0.5	0.25	0.5	
As		0.5	0.25	
G			0.25	

**Fig. 43** Transition matrix for harmonic state generation.

This transition matrix can also be written as: C C B, B B As As C, As As B B G, G C (or 1 1 2, 2 2 3 3 1, 3 3 2 2 4, 4 1, in figure 45, by giving the states a number). In this formulation of groups of states separated by a comma, the first state of every group denotes the actual state followed by its possible successors, where the number of times a state shows up determines its probability. When programming in Puredata this formulation makes it easier to implement a Markov process.



**Fig. 44** Markov Model as state transition graph.

Given this basic harmonic skeleton, the next step is to generate the note values/ pitch sequences to complete the HMM. The simplest way to do that would be to choose randomized or weighted values out of a list given by the harmonic state. Another possibility, which I will realize in this case, is to use another Markov process for that task. This should bring some structure in the surface layer by reducing the arbitrariness of notes in direct successions<sup>75</sup>. So the basic harmonic skeleton is produced by a first MM (figure 44) and another MM will determine the pitch values.

The first MM (harmonic skeleton, the hidden one) will be a stable model for generating a movement over a longer period of time. The second MM (for the note values) should be more volatile and change over time to produce more change and surprise in the resulting sequence of note values. For this purpose, I will define the corresponding transition matrix by generating the training sequence by a stochastic process. The resulting MM can be employed for a period of time (which will appear more stable to the listener) and then be changed by generating a new transition matrix (which will appear as a major change). In this example, the beginning of a new 'round' of harmonic states marks the point in time when a new MM for the pitch values will be generated<sup>76</sup>.

**Cross-relations:** When using a large-scale structure generator like shown here it can be an interesting approach to use its output also in other fields of the generated musical piece, like rhythm, effects, etc. The same Markov process can be used to form another HMM which is linked to the rhythmic dynamic in some way. The goal of such a cross-relation is to emphasize the wholeness of the resulting structure by using a technique of parallel evolution (see chapter 2.3.2) between the melodic and the rhythmic dynamic<sup>77</sup>.

75 Rens Bod (Bod 2001) shows the importance of direct succession within the layer of perceived pitches by testing different parsing strategies on a large folk-song collection. Apparently, the perceived structure can be captured much better when introducing a Markov Model reflecting the probability of the succession of note values – which in this case is taken out of a large set of folk songs ('training set').

76 This approach is documented as an audio example on: [werkstatt.hotglue.me/sound](http://werkstatt.hotglue.me/sound) (Jan 2021).

77 This is exactly how the rhythmic background, the bass line, and the melody line are related in the audio example 'Hidden Markov Chain' on: [werkstatt.hotglue.me/sound](http://werkstatt.hotglue.me/sound) (Jan 2021).

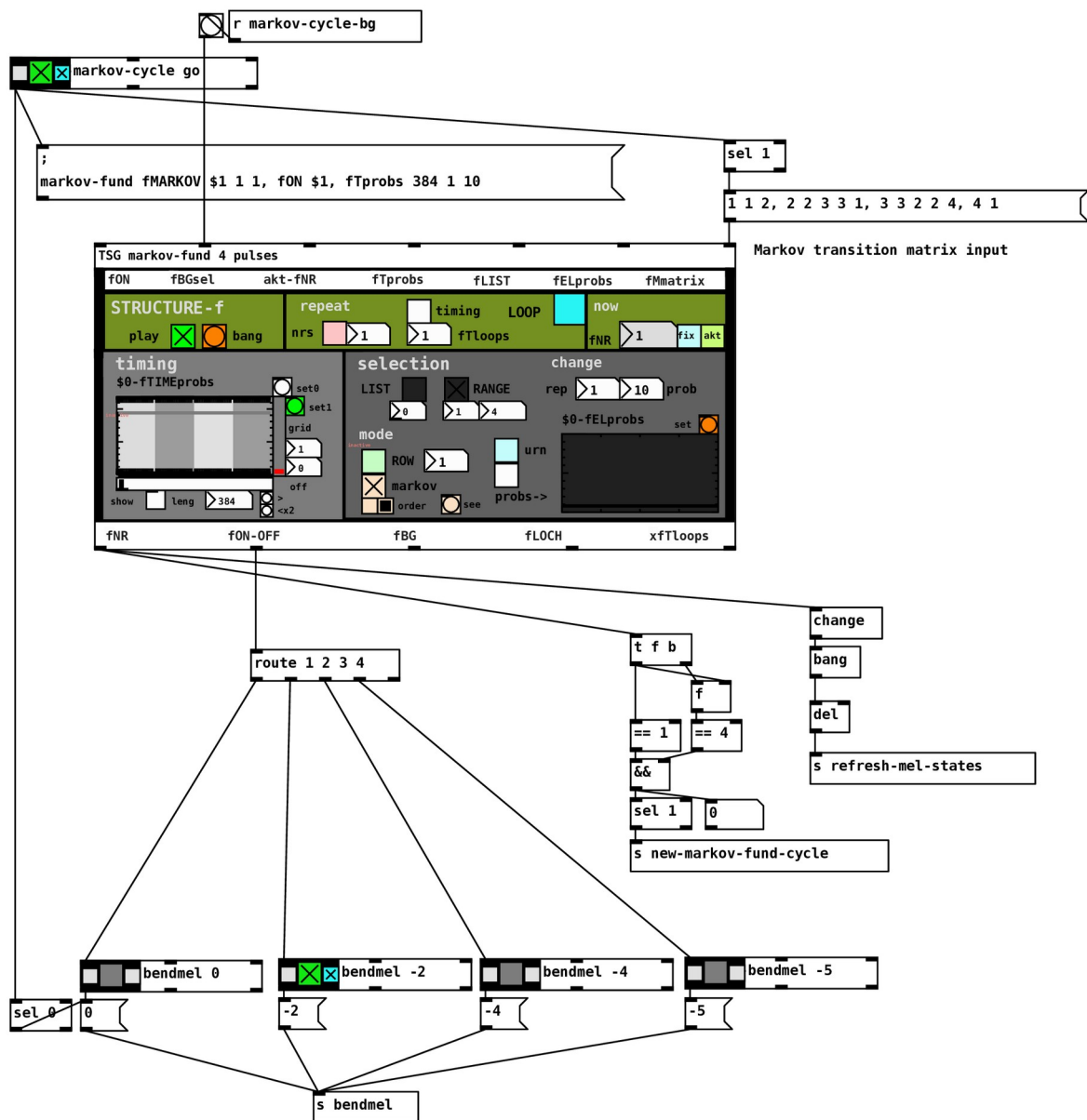


Fig. 45 The HMM implemented in Puredata.

Figure 45 shows the corresponding large-scale structure generator implemented in Puredata: A TSG object generates the states of the harmonic MM. The four possible states are represented as 'losta' objects named 'bendmel', which will be switched according to the MM. The transition possibilities are introduced by the message sequence to the upper right. The 'send refresh-mel-states' marks the points in time when the harmonic state is changed. The 'send new-markov-fund-cycle' marks the points in time when the Markov process comes back to its starting state. The 'losta' objects communicate the active state within the generative system and can so be used to build HMMs for different purposes.



## 2.4 The graphical user interface (GUI)

As I mentioned at the beginning of chapter 2, one of the challenges was to develop a setup that could be used both for human improvisation as well as for machine improvisation by the creation and running of generative production mechanisms or systems. I used the term 'algorithmic improvisation' to describe this open interaction between algorithmic structure generation and live intervention within one sound-producing system (see chapter 2.1). The key to creating a system that meets these criteria on a PC or laptop is, of course, the graphical user interface (GUI). In chapter 2.5 we will see the generative sound sources having their own GUI allowing the user to set different parameters, and to start or stop the sound production. In this chapter, I will describe the GUI for the whole generative sequencing system its implementation in Puredata. When considering the GUI, the programming environment is quite decisive as it offers specific methods of interacting with a production system. Being a graphical programming language, Puredata makes it relatively easy to set up graphical representations and control mechanisms for the running processes.

When I speak about the GUI here I am referring to a visible control structure built for operating a generative sequencing system by user intervention. This control center is set up in a Pd-patch where the most important parameters can be controlled. It should therefore provide control over the following areas:

- **Audio mix control:** Control over audio parameters (volume, panning, etc.). Mixing and routing audio signals from groups of generative sound sources to the sound output.
- **Sound source control:** Control over the involved generative sound sources and their previously defined play-modes.
- **Automation control:** Control over generative mechanisms (large-scale structure generators in figure 22) like the grouping and arranging of generative sound sources and their play-modes and parameters.

Figure 46 shows a schematic diagram of the GUI and its different areas of control. Figure 47 shows a possible implementation in Puredata. Here we see mixing consoles for the different audio groups which are bundled to form the master track (yellow). The control over the sound sources and the automated structuring mechanisms or (large-scale structure generators) is organized with the help of 'losta' objects which can be switched in the GUI.

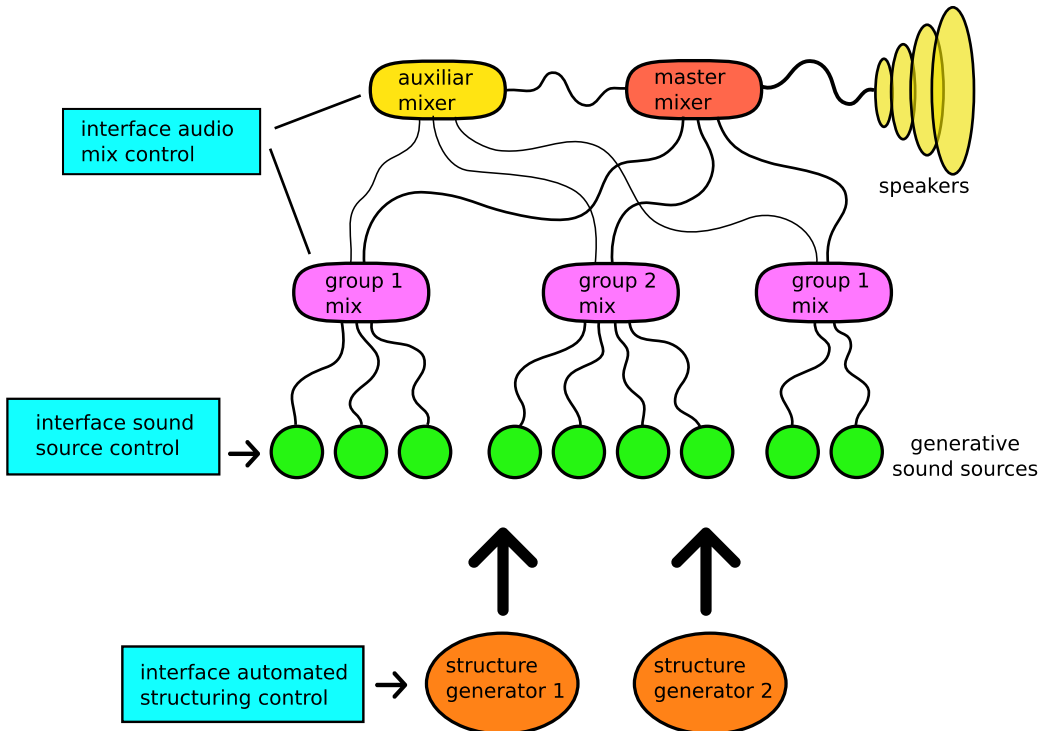


Fig. 46 Schematic model of the GUI for controlling a generative sequencing system.

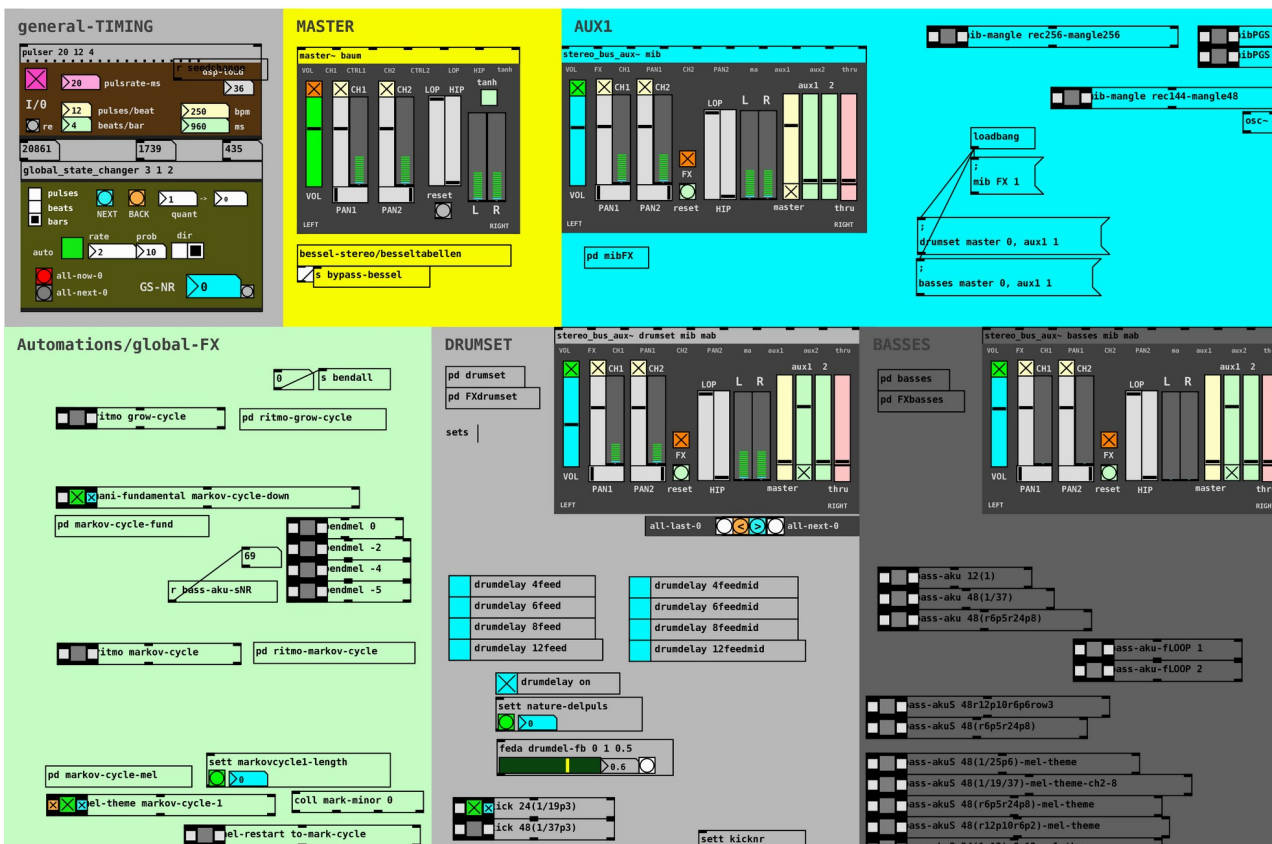


Fig. 47 Example of a GUI implemented in Puredata.

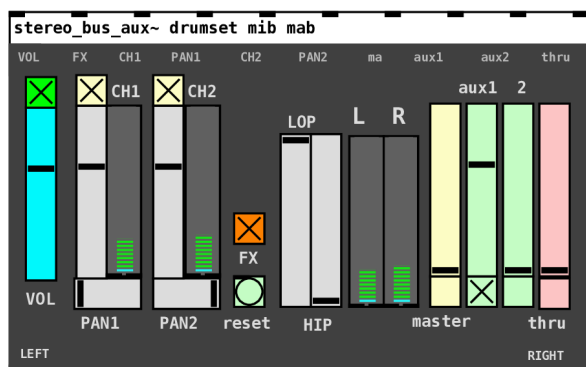
## 2.4.1 The mixing interface

In order to run several sequencers or sound sources, a mixer setup is required that allows to form different groups, to place effects on individual or multiple sound sources, to influence the resulting mix as a whole, etc. The mixer setup itself is not at the core of the generative sequencing processes but it allows to expand these processes over a wider range of sound characteristics and to place structuring interventions at different layers of the system. Sound sources can be grouped to form perceptible musical streams by bundling them into one channel and applying effects or filters on it. Effects and filters themselves can be connected with other timed processes or receive their own temporal structuring. In any case, the mixer setup is a central aspect of the practical work in sound production.

The system I use for mixing and controlling the different audio signals present in a sequencing setup corresponds roughly to a mixer or mixing console: Several tracks are combined into a master track which is connected to an amplifier and speakers via a suitable audio interface. There is the possibility to send the individual tracks to an auxiliary channel to create groups on which effects can be placed and the like. In figure 44 a possible mixing arrangement is shown. In this case, the vertical areas represent a group of sound sources that are bundled to form an audio group (channel or track). Audio parameters like volume, panning, and routing can be set in the stereo mixing consoles at the top of every group. All the audio groups together are summed up to the master channel which allows controlling the important audio parameters of the entire mix before going to the amplifier.

### The stereo setup

Depending on the application of the generative sequencing system it is preferable to work with different numbers of audio channels and spatial distributions of speakers. Figure 47 shows a standard stereo setup for two audio channels.

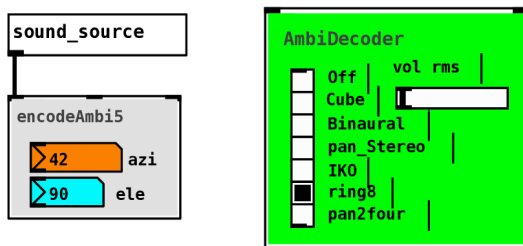


The 'stereo-bus-aux' object is designed like a stereo mixing console for controlling one or several sound sources bundled to a channel. The interface consists of volume control, panning control, a low-pass and a high-pass filter, and a toggle to enable effects. The output audio signal can be routed to the master track, two auxiliary tracks, and the signal outlets of the object in Pd ('thru').

*Fig. 48* The 'stereo\_bus\_aux'.

## The Ambisonic setup

When creating immersive sound installations it is suitable to use more than two speakers for distributing sound events around the listener with a higher spatial resolution. The 'Ambisonic' format<sup>78</sup> is an audio format that makes it possible to design pieces with detailed spatial information of the included sound events and to flexibly organize sound installations with multiple speakers. The individual sounds or sound sources can be located and moved in the room with the indication of azimuth and elevation. It is also possible to change the position of the sounds automatically and to use causal chains that connect certain events with certain positions or movement patterns. In the generative system described here, the sound sources can be distributed and moved in space by defining their coordinates via 'azimuth' and 'elevation' (figure 49). Controlling the position of the sound sources opens a further dimension, where structuring and generative mechanisms can be applied.



The 'encodeAmbi5' object allows situating a sound source in space. The 'AmbiDecoder' maps the result to the present output system (i.e. the distribution of speakers).

**Fig. 49** The 'encodeAmbi5' and the 'AmbiDecoder'.

<sup>78</sup> See (Zotter/Frank 2019) for more information on the Ambisonic format and the spatial distribution of audio sources.

## 2.4.2 A GUI for improvisation with local states and structure generators

As shown in chapter 2.2 the use of local states (or 'losta' objects in Pd) provides a simple communication medium between the different layers of the generative sequencing system – it connects the user interface and large-scale structure generators with the generative sound sources (see figure 25). One of the reasons for introducing the local states was the necessity of dealing with multiple generative sound sources in one system without losing the overview in the GUI. The local states can represent the play-modes of the generative sound sources present in the system. They can be displayed and switched on one screen without displaying all the corresponding sound sources, like in figure 47, where the 'losta' objects are placed below the stereo mixing boxes of the audio groups. This way it is possible to improvise with the play-modes by switching them on and off manually or automatically with the help of a superordinate structure generator. As we have seen in chapter 2.2 the local states can be used to group various play-modes and form tree-like structures while still be placed and operated within the same GUI. This makes it possible to run automatic structure generators operating on the sound-sources as the user, who still can intervene or improvise in parallel.

### Global states and local states

When using a generative sequencing system for live improvisation with I found it necessary to be able to switch several local states at the same time and being able to go back to the last distribution of active states (so to switch from one distribution to another one and back again). After experimenting with different setups and mechanisms for grouping and organizing the local states I ended up using 'global state numbers' that describe the system in terms of active local states at a given point in time. The mechanism consists in marking the local states to be active or inactive in the next global state and then increment the global state number. That's what the additional toggles of the 'losta' object are designed for (see figure 30). The main toggle (in the middle) shows the current state of the 'losta' – ON or OFF. The smaller toggle to the right sets the state of the 'losta' after increasing the global state number. The smaller toggle to the left sets the state of the 'losta' after decreasing the global state number. So if the 'next' toggle is ON, the 'losta' will be ON after switching the system to the next 'global state', when it is OFF it will be OFF.

The 'losta' remembers its state at all past global state numbers and recalls it when going back to a global state number that has already occurred in an improvisation. Like this it is possible to reverse one or several steps, recalling the distribution of active local states at a given global state number. That's also where the 'all-next-off' and 'all-last-off' buttons of the 'global state changer' come into play – they switch off the 'next' or 'last' toggles off all involved 'losta' objects<sup>79</sup>.

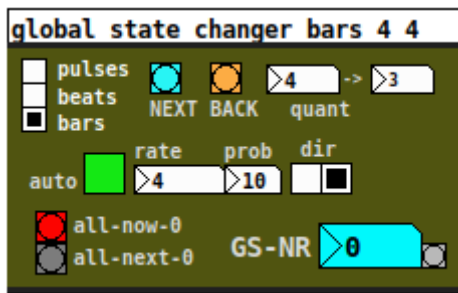
---

<sup>79</sup> To give a simple example of what I use this mechanism for: If several sequencers are playing and I want only the base drum to continue, and then, after a certain time, go back to the former state, I simply would switch off all 'next' toggles with the 'all-next-off' button in the 'global state changer'. Then I would switch the 'next' toggle of the bass

global state number:	to sound source A:	to sound source B:	to sound source C:
1	local state A a		
2	local state A b	local state B d	
3	local state A a	local state B e	local state C g
4	local state A a	local state B d	

**Fig. 50** A table showing increasing global state numbers and corresponding active local states.

Figure 50 shows a possible sequence of global state numbers in relation to the active local states in a system with three sound sources (A, B, C). The local states specify the active play-mode of the corresponding sound source. A table like this describes the evolution of the generative system in terms of active local states (and associated play-modes) at a given global state number.



**Fig. 51** The 'global state changer' object in Puredata.

The Pd object 'global state changer' (figure 51, in figure 47 in the upper left corner) sets a global state number and communicates this number to the 'local states'. The current 'global state number' shows up in the light blue 'GS-NR' number box. The GUI allows to switch between 'pulses', 'bars' or 'beats' (emitted by the 'pulser' object) as timing reference. The 'NEXT' and 'BACK' buttons allow to switch between 'global states' manually, with the quantization specified in the 'quant' number box. This quantization is important, because in many cases it is preferable to change the 'global-state' and the depending 'local states' at a certain point in time, for example at the beginning of the next bar. The smaller number box to the right shows the current position in the quantization grid.

The green 'auto' toggle switches the change of the 'global states' to automatic. In this case the 'global state number' will increase or decrease automatically with the specified rate. The 'prob' number box sets the probability for changes to happen (10 is a probability of 1) and the 'dir' switch allows to set the direction to right (forward, increases the 'global state number') or left (backward, decreases the 'global state number'). The buttons 'all-next-off' and 'all-now-off' refer to the 'local states'. The 'local-states' can be switched all together by these buttons which is particularly useful in live improvisation.

---

drum on again, before switching to the next 'global state' by pressing the 'NEXT' button of the 'global state changer'. Finally, after the desired time with only the bass drum playing I would press the 'BACK' button of the 'global state changer' to switch everything back to the former state.

## 2.5 The generative sound sources

In this chapter, I will present some generative sound sources as I use them in the generative sequencing system described here. The generative sound sources are in principle structure generators combined with sound production mechanisms and a graphical user interface. They can be connected to other (larger scale) structure generators via the communication medium (local states, see chapter 2.2). They can also be seen as a form of virtual instruments and they can be characterized by the following common features:

- The use of a GUI (Graphical User Interface): The sequencers presented have a graphical user interface and can thus be operated in real-time on the screen of a laptop or PC using the mouse. This workflow is particularly suitable for live improvisation.
- The possibility of automated control: The sequencers presented can be controlled via "messages" in Puredata. Thus, automation can be designed, such as the recalling and relating of certain settings or events. This approach is particularly suitable for sound installations or programming generative systems.
- They are generative in some kind and dispose of methods for randomization and the limiting of randomness. The aim is to create and structure relationships between the sound elements used and their possible variations.
- They include probabilistic modes of operation: They are probabilistic in the sense that essential variables are entered as probabilities.
- They are open for mutual combinations: Any number of sequencers can be operated simultaneously and interconnected in different ways.

In the following I will describe two generative sound sources I implemented in Pd and which I use frequently: The 'pitch\_group\_sequencer' (PGS) and the 'loop\_slice\_mangler' (LSM). However, in my practical realizations, I also used other, mostly simpler, generative sound sources. Depending on the application area different stochastic and generative methods can be applied to produce structured sequences. Thereupon different mapping strategies can be used to map symbolic or numeric content to the production of sound<sup>80</sup>.

---

80 More information on the use of stochastic methods for the production of sound and the mapping of symbolic values to sound production mechanisms can be found in (Farnell 2010).

## 2.5.1 The 'Pitch Group Sequencer' (PGS): A two-dimensional probabilistic sequencer

The PGS (Pitch Group Sequencer) is a sample playback sequencer. It works by selecting samples within a group of samples (i.e. sound files), selecting the playback speed (or 'pitch'), and selecting the points in time for starting the playback.

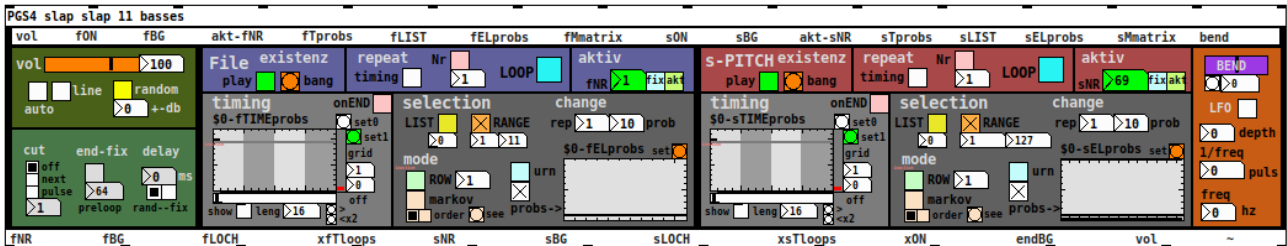


Fig. 52 The PGS (Pitch Group Sequencer).

The PGS consists of two sequencers running in parallel, which correspond to the TSG ('Timed Structure Generator') described in chapter 2.3. One TSG determines the selection of the sample within the defined group, the other TSG determines the reading speed of the sample (and thus the pitch). So the PGS produces two numbers at certain points in time representing the sample number and a pitch value (or playback speed) which triggers a sample playback mechanism. The selection of these numbers is based on various types of weighted randomness and the setting of probabilities (the 'selection modes'). In practice, I have mainly used this sequencer to play and vary samples for creating rhythmic and melodic patterns.

### The volume control:

To adjust the volume, a slider is used which can be adjusted manually or set by 'message'. Furthermore, it is possible to automate the volume in a loop, i.e. to set values that are repeated in a certain period of time. The automation can be discontinuous or continuous ('auto' or 'line'). The random button varies the volume randomly for every triggered sample in the entered range.

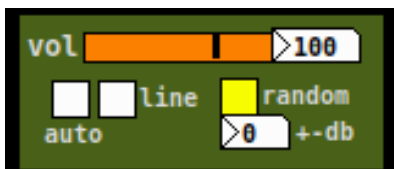


Fig. 53 The volume control.



### Sample overlay and timing preset:

The PGS can reproduce a maximum of 10 samples simultaneously. It is often useful to cut off a sample that is still running when a new one is triggered. This is achieved by setting the 'overlap' number to a lower value, specifying the maximum number of parallel playbacks. It is also possible to limit the samples played back to a fixed duration.

On the right-hand side one can also define a preset regarding timing:

- 'endfix' sets not the beginning but the end of each sample to the selected point in time.
- 'delay' delays the trigger time by a fixed or random value.

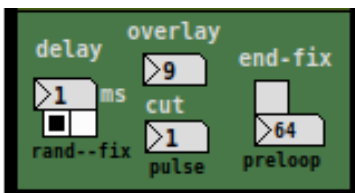


Fig. 54 The overlay and timing adjustment.

### The 'File' and 'Speed' sequencers:

The timing and selection mechanism of the file and speed sequencer sections are working in the same way as the TSG described in chapter 2.3. In the PGS two TSG are combined in such a way that variations in the timing and the selection within a group of sound files and their playback speed can be created by probabilistic constraints.

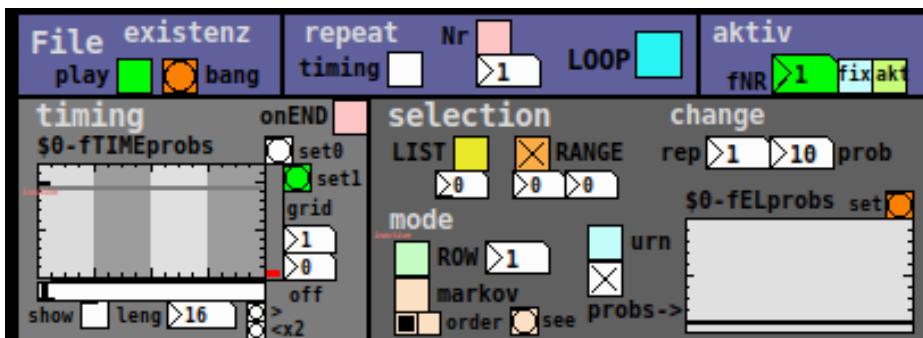


Fig. 55 The file sequencing part of the PGS.

### The 'BEND' section:

This section at the right side of the GUI allows to 'stretch' and 'squeeze' the played samples in real-time by adapting the playback speed. It has an integrated low-frequency oscillator that can be timed in pulses to match the system timing.

## Describing the PGS with a probabilistic grammar model:

### Restricting randomness:

The approach of probabilistic sequencing like it is done by the PGS is characterized by randomness and its limitation (by excluding possibilities) and weighting (by setting probabilities). The constraints generated by these restrictions determine the form and structure of the resulting sound sequence. The restrictions of randomness employed here can be described and systematized in different ways. They can be simple limitations that reduce the range of possibilities, the weighting of the existing possibilities, or more complex strategies based on the setting of internal dependencies and rules concerning timing, loop length, etc.

The use of probability is a way of restricting randomness by weighing different possibilities. In the PGS, both the timing and the characteristics of an event can be expressed by probabilities, which are shown and set in a visual array or via a message or the GUI. In figure 55, the left array named 'fTIMEprobs' shows the probability of an event at all time slots in the loop. The right array, named 'fELprobs' shows the probability of each event to be selected.

A generative grammar can be notated as quadruple  $(V, V_t, S, P)$  in the following form:

$V$  a finite set of non-terminal symbols

$V_t$  a finite set of terminal symbols

$S$  starting symbol of  $V$  without  $V_t$

$P$  a set of rewriting or production rules:  $a \rightarrow b$ , where  $a \in V^+$  and  $b \in V^*$ <sup>81</sup>

The rules by which a 'sentence' or loop is formed in a sequencer can be formulated as generative grammar and can be deterministic or probabilistic.

Since such a consideration of sentence formation through generative grammars is revealing when designing and using the PGS, I will describe the grammar of the PGS in a little more detail and give a few examples. First of all, a general grammar can be formulated that includes all possible sentences that can be formed by the PGS. This corresponds to all the possibilities that can occur in a completely open, i.e. randomized, playing mode.

In a second step, further grammatical rules can be formulated that correspond to specific playing instructions. In doing so, the general grammar of the PGS (all possibilities) is further restricted to a grammar with more specific rewriting rules producing a less randomized result.

---

81  $V^*$  is the so called 'Kleene closure' over  $V$  (all possible expressions within a grammar).  
 $V^+$  is the positive Closure over  $V$ :  $V^*$  without empty word string.

### **Definition of a non-deterministic generative grammar of the PGS:**

A Loop (L) consists of sound events. In the PGS, three variables must be determined for each sound event (K) (see above)

- $t$  = the point in time within the loop
- $s$  = the number of the sample within the group of samples (sample number)
- $p$  = the pitch (or reading speed)

Any sound event (K) consists of these three data:  $t$ ,  $s$  and  $p$  – time, sample number and pitch:

$$K = (t,s,p)$$

The variables  $t$ ,  $s$ , and  $p$  will be replaced by concrete numerical values in the last rewriting step of the grammar. The choice of numerical values can be restricted and structured in various ways – but first I will define the grammar for all possible expressions that the PGS can produce.

When writing a loop in the PGS we start with the loop (L), replace it with a number of sound events (K) at some points in time ( $t$ ) in the first rewriting step. In the second step, we replace the sound event with the sample-number  $s$  and the pitch  $p$ . L (Loop) is, therefore, the start symbol. Furthermore, K,  $t$ ,  $p$ ,  $s$  are used as non-terminal symbols. The terminal symbols used are integers which are finally interpreted as the instructions for reproducing a sample.

Considering the construction details of the PGS we can make the following assumptions:

- For  $t$ :  $1 \leq t \leq T$  (i.e. the selectable points in time located within the total length of the loop (T)).
- For  $s$ :  $1 \leq s \leq S$  (i.e. the sample number is located within the total number of selectable samples(S))
- For  $p$ :  $-127 \leq p \leq 127$  (i.e. the playback speed (or pitch) ( $p$ ) is located within the note values -127 to 127, which are midi note values interpreted as a positive and negative sample reading speed)

The resulting definition of a general generative grammar for the PGS is:

V L, K,  $t$ ,  $s$ ,  $p$ , e (e = empty)

$V_t$  integer numbers, e

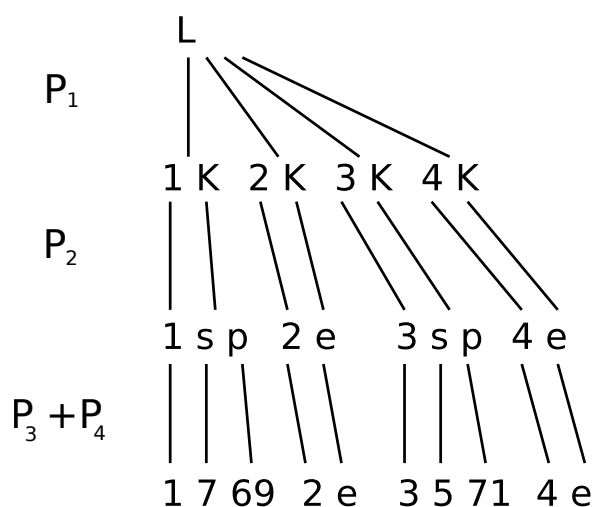
S L

$P_1$   $L \rightarrow 1 K \dots t K \dots T K$  ( $1 \leq t \leq T$ , T is the length of the loop to define)

$P_2$   $K \rightarrow s p \mid e$  (K is replaced by sample number and pitch, or empty)

$P_3$   $s \rightarrow$  integer from 1 to S (sample number)

$P_4$   $p \rightarrow$  integer from -127 to 127 (pitch value)



**Fig. 56** Visualization of a grammar tree for the PGS. In this example a simple phrase.

Figure 56 shows an example phrase for a loop of length four ( $T=4$ ). On the first time slot the sample no. 7 is reproduced with a reading speed of 69 (original pitch). Time slots two and four are empty, while on time slot three sample no. 5 is played with speed 71. The selection of the terminal numbers and the decision of which time slots stay empty were taken randomly in this case. How the structuring of a phrase through the definition of rules or constraints can be done, will be shown in the following.

**Grammar-specifications: Some examples for probabilistic or stochastic<sup>82</sup> generation:**

For the creation of structured patterns by determining the sound events in the respective loop, different rules or algorithms can be applied. These rules correspond to grammar-specifications, which represent the playing instructions for the PGS by allowing only certain elements at certain points in time to be selected. The PGS allows to If only one possible sentence remains, it would be a deterministic grammar. The examples discussed here usually deal with the assigning of numerical values  $t$ ,  $s$ , and  $p$  while the main structure of the PGS-grammar remains as described above.

**'set time probabilities':**

This rule can be formulated as follows:

*set-tPr* [ $t_1(Pr_1)$ ,  $t_2(Pr_2)$ , ...  $t_n(Pr_n)$ ]: Set the probability for an event at a certain point in time  $t_n$  to  $Pr_n$ . Set the probabilities for all  $t$  which are not mentioned to 0.

This allows sets the probability of an event at certain points in the loop to a certain value. At the same time, the probability of all other points in the loop is set to zero. In the PGS the probabilities

<sup>82</sup> see (Lindenmayer / Prusinkiewicz 2004, 28 f) for an explication of stochastic production systems

can be set by a message and are then displayed graphically in arrays (see chapter 2.3). When defining a loop the points in time get assigned different probabilities for either an event or not. For this purpose the following grammatical rules will be defined:

- V L, K, Pr, s, p, e (e = empty)
- V<sub>t</sub> integer numbers
- S L
- P<sub>1</sub> L --> 1 Pr<sub>1</sub> K 2 Pr<sub>1</sub> K ... T Pr<sub>T</sub> K (T is the length of the loop to define, Pr<sub>t</sub> is specified in the definition of the probability array)
- P<sub>2</sub> Pr<sub>t</sub> K --> s p – to a probability of Pr<sub>t</sub>  
or Pr<sub>t</sub> K --> e – to a probability of (1-Pr<sub>t</sub>) (K is replaced by s p, or e, depending on time specific probability)
- P<sub>3</sub> s --> integer from 1 to S (sample number)
- P<sub>4</sub> p --> integer from -127 to 127 (pitch value)

In figure 57, an example of a grammar tree using the 'set-tPr' rule specification is shown, assuming that the number of time slots in a loop is eight (T=8):

'set-tPr [1/10, 3/5, 5/5, 7/5]': Set the probability for an event at the first time slot in the loop to 100%, for the third, fifth and seventh time slot to 50 %. No events happen at any other point in time of the loop.

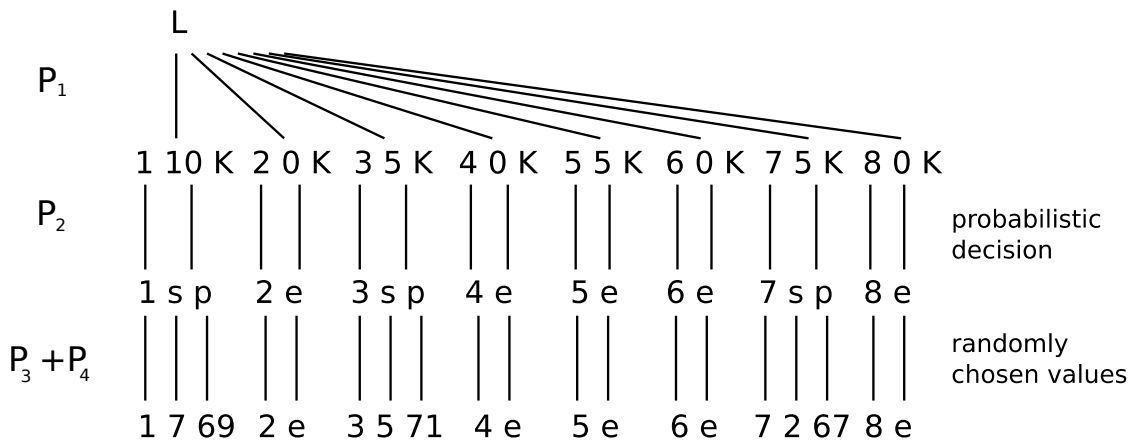


Fig. 57 Visualization of a grammar tree example with a probabilistic rule.

**'set pitch-range':**

A rule for restricting the set of possible pitch values:

sPR [a/b]: Set the possible playback speeds from a to b:  $a < p < b$

Example: sPR [57/81]: Limit the range of playback speeds to 2 octaves. In the PGS-grammar model, this means that in the last rewriting step for p only values between 57 and 81 are possible.

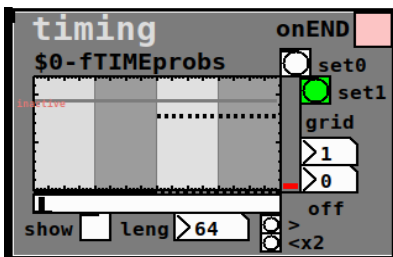
**'set sub-loop probability':**

A rule for setting periodic probability values:

*sSP [ta/tz/Tl/P]: Start at time ta with a sub-loop of length Tl and set the probability of an event at the first point of time of the sub-loop to P; repeat this procedure until time tz is reached.*

An example is shown in figure 54:

sSP [33/64/2/8]: Set the probability of an event at the time points 33,35,37,...,63 to 80%.



**Fig. 58** The probability array of the PGS after setting a 'sub-loop probability'.

## 2.5.2 The 'Loop Slice Mangler' (LSM): A looper and beat-slicer for generative manipulation of sound loops

The 'Loop Slice Mangler' or LSM is a looper and beat slicer that allows various forms of fragmentation, rearrangement, and variation of loops and samples.

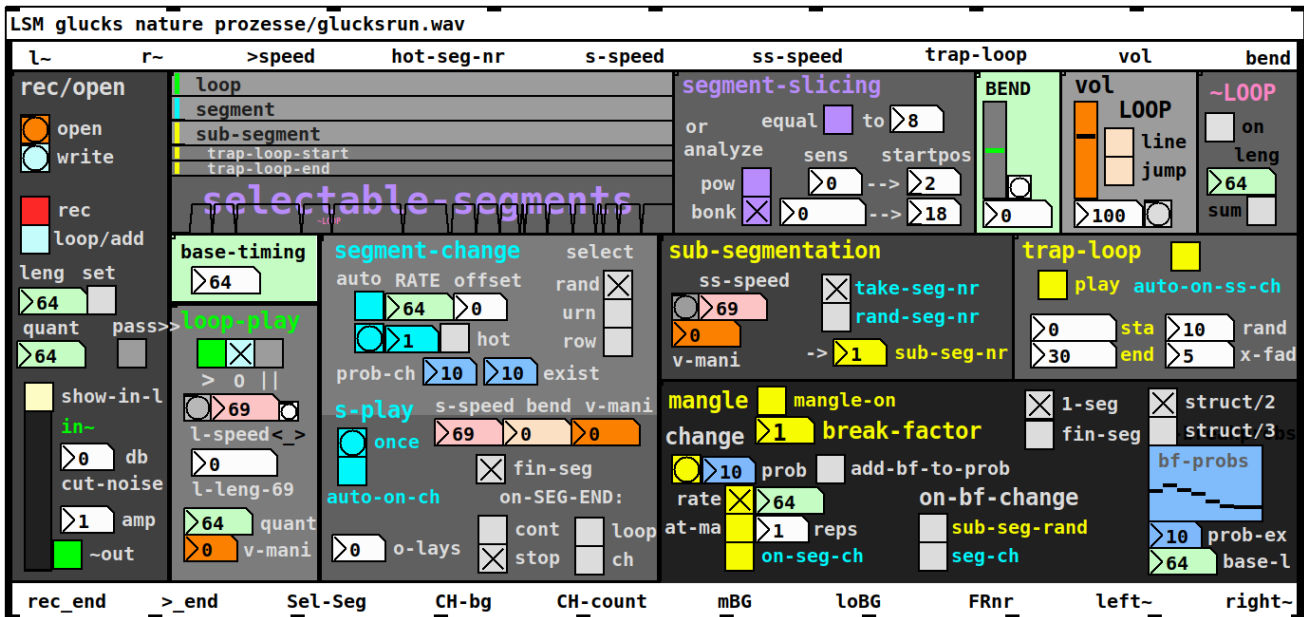


Fig. 59 The GUI of the LSM.

The LSM works with a sound loop (i.e. a sound file played in a loop). The sound file used can be loaded from a storage medium or recorded live. It forms the basic loop, which can be reproduced, changed in speed and duration, or sliced into smaller pieces and re-arranged in various ways by the LSM. This process of slicing, varying, and rearranging is referred to by the term 'mangle'. As the loop is segmented into smaller pieces it is preferable to use sound samples containing various distinct elements or attacks.

The underlying idea of generative beat slicing is to develop a table look-up software tool that fits sound files (i.e. the tables to lookup) into a temporal grid and is also able to break them down into smaller units, slicing the sound file at desired points in time (beats or attacks). The smaller units could be organized into patterns and reproduced within the rhythmic grid. To enable the generation of variable patterns random variables can be placed at various places in the process<sup>83</sup>. In the following, I will briefly outline the most important technical aspects in general and then describe the implementation of the LSM in Puredata.

83 See <https://www.soundonsound.com/techniques/beat-slicing-masterclass> (Jan 2021) for an introduction to beat slicing and an overview of beat slicing software.

### **Fitting of a sound file into a given rhythmic grid:**

To use a sound file as a loop within a piece of music, it is necessary to be able to adjust the length of the loop flexibly. This is done by changing the length of the sound file used, which can be stretched or compressed to the appropriate length in time-related to the system timing. It should be noted that a change in length is associated with a change in pitch.

### **Slicing a sound loop into smaller segments:**

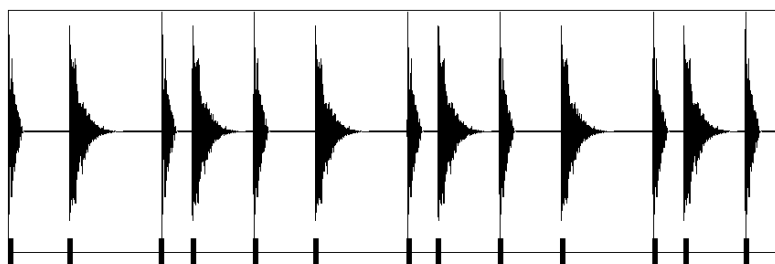
The slicing is done by setting and resetting starting points for the lookup mechanism within the loop. The selection of the starting points is synonymous with the selection of beats or segments within the original sound loop.

To set potential starting points, there are different possibilities: First, dividing a loop into pieces of equal length: To set the starting points according to the quarter or eighth notes, the original loop must be divided into four or eight parts of equal length. The starting points of the quarter notes are located exactly after a quarter of the total length (see figure 60). Therefore the original loop should be structured in such a way that the timing matches the slicing. For example, a loop in three-quarter time would have to be divided into three and not four pieces.

Second, the starting points can be set according to the beats, transients, or attacks actually present in a sound loop. For this, a sort of beat-finding algorithm must be implemented. Such a mechanism analyses the loaded sound file to set possible starting points (see Fig. 61). Detecting transients or attacks is also an interesting feature for the use of more versatile sound material that is not necessarily based on beats (like field recordings, human speech, etc.)



*Fig. 60 Partition of a bar in traditional notation.*



*Fig. 61 Possible starting points correspond to the actual attacks in a loop.*



### The selection of slices – timing and sequence:

An important feature that a generative beat slicer should have is the ability to reproduce the beats or segments of the original loop with independent timing. Furthermore, it should be possible to select the segments within the loop in a different order.

This requires a tool for timing and triggering the segments extracted from a loop. To increase the generative capabilities this tool should have the ability to set probability values for timing and incidence. It would also be advantageous to be able to influence the repetition and selection behavior, for example, repeat the last selected beat or set the selection behavior to random or row. An interesting stylistic element is the fast repetition of single beats in a fraction of the bar length.

Figure 62 shows an example of a sound-file loaded into a generative beat slicer: First, the sound file has to be fit in a time grid, in the example shown it is fit to one bar of four notes. Second, the segments which have been detected in the sound file can be changed in timing and order. Third, the segments can be repeated within a shorter period of time. The name of the LSM (Loop-Slice-Mangler) refers to these three sequencing or play-modes: 'Loop' a sound file in chosen length, 'slice' it into segments, 'mangle' it by repeating its segments in short fractions of time.

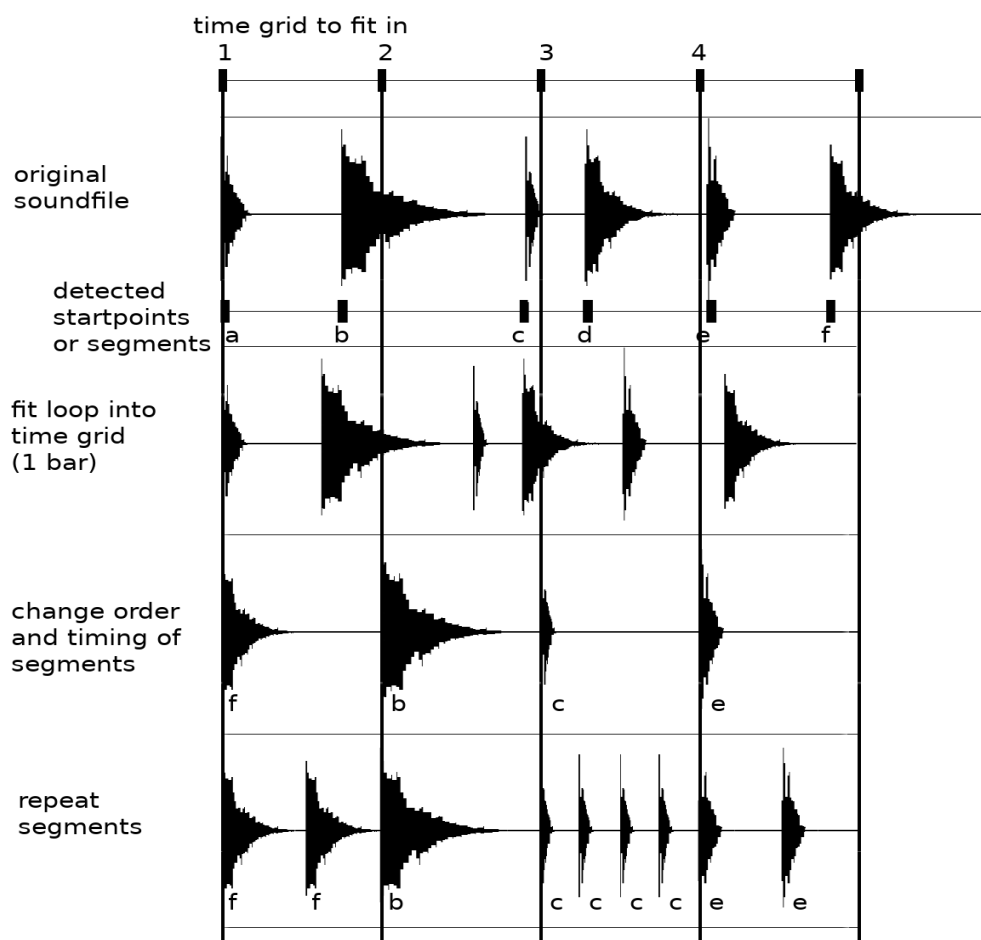


Fig. 62 Schematic representation of the rearrangement of a sound-file as carried out in the LSM.

## Implementation in Puredata

### Record, open, and analyze a sound file:

Figure 63 shows the part of the LSM GUI for loading and recording sound-files. This part of the LSM allows to load or record sound-files of any length. With the 'open' button a saved sound-file can be recalled. Alternatively, this can be done automatically by a message when opening a patch.

The red 'rec' toggle is for recording a sound file. With the 'set' toggle and the 'leng' number box, it is possible to determine the length of the file to be recorded in advance. In this case, the recording will be stopped after the specified time. The length is specified in 'pulses' (see chapter 2.3). In this case, the recording is stopped after the specified time has elapsed. The length is specified in 'pulses'. If 'loop/add' is checked, a loop is recorded in the specified length and overdubbed until 'rec' is deactivated. This allows sound files to be recorded in several layers. If a quantization of the start and end times of the recording is desired, the desired value can be entered in the 'quant' number box. In the lower section, the incoming audio signal can be displayed and amplified. The incoming signal can be recorded as a single-channel (mono) or two-channel (stereo) track.

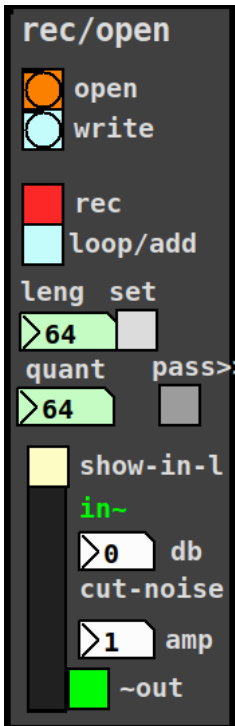


Fig. 63 The 'rec/load' section.

### The timeline and the selection of starting points:

The LSM can run three play-modes simultaneously, so there are three timelines in the GUI (figure 64): The loop timeline (top, green), the segment timeline (light blue, center), and the sub-segment timeline (bottom, yellow). The colored positions in the timelines indicate the respective position of the loop player, segment player, and sub-segment player. In practice, a sound file can firstly be played in a loop, secondly played as individual segments, and thirdly fragmented into even shorter sub-segments. All these sound elements can be reproduced according to the probabilistic timing structure offered by the LSM. The three play-modes can be activated simultaneously and at different playback speeds and volumes.

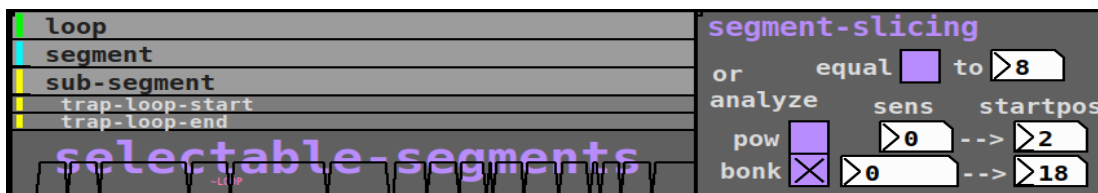


Fig. 64 The LSM timeline and the 'selectable-segments'.

The 'selectable segments' shown at the bottom of the timeline are mainly for orientation purposes and show into which segments the sound file is sliced. The slicing into segments can be done by dividing the sound file into pieces of equal length (with the 'equal' button and setting the number of segments desired) or by detecting transients ('pow' analyzes the sound pressure finding points in time just before the signal gets louder, and 'bonk' uses a Pd object called 'bonk' for finding transients<sup>84</sup>).

### Playing a loop:

The controls for play (>), loop (o), and pause (II) are on top. If play is activated the LSM starts playing the loaded sound file according to the quantization entered in the 'quant' number box. The pink number box indicates the current playback speed. If the playback speed is changed, the length of the loop changes accordingly. In the number box 'l-leng-69' the loop length is displayed at the original playback speed (=69). If this value is changed, the sound-file will be stretched or compressed. This function is mainly used to integrate a loop into the rhythmic grid of a sound piece or an improvisation. The number box 'v-mani' allows manipulating the volume of the loop-player without influencing the other play-modes.

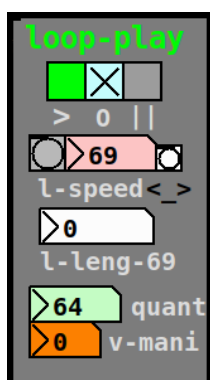


Fig. 65 The 'loop-play' section of the LSM.

### Playing segments within a loop:

The segment play section mainly has the function to reproduce segments within the sound loop loaded in the LSM. The upper part of the GUI controls the selection of the segments. The timing for the reproduction can be set to automatic triggering with the 'auto' toggle with a specified rate and offset regarding global timing (see chapter 2.1). The current segment is shown in the light blue number box.

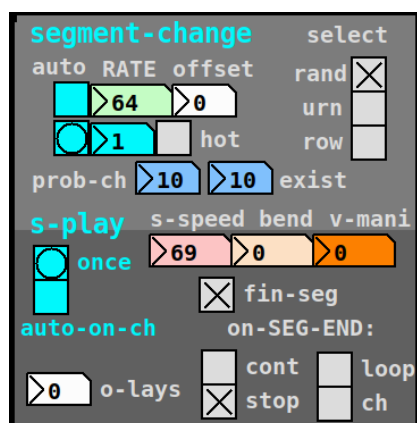


Fig. 66 The 'segment-change' section.

84 See the Puredata help file bonk~help.pd, included in Pd 0.50.0: 'The Bonk object takes an audio signal input and looks for "attacks" defined as sharp changes in the spectral envelope of the incoming sound.'

The bang button to the left changes the segment immediately. The probability of changing the segment is specified in the 'prob-ex' (probability of existing i.e. triggering a change) and 'prob-ch' (probability of changing the segment number). For more sophisticated selection modes the LSM can be combined with the PGS and use its generative number selection features (see chapter 2.5.1)

The lower part (below the blue 's-play' lettering) controls the reproduction of the different segments: The 'once' bang starts the reproduction of the chosen segment immediately. The 'auto-on-ch' toggle starts the reproduction of a segment automatically if the segment changes. In this case, the segment change controls the timing. To the right, the reproduction speed can be set in the 's-speed' box. This reproduction speed is set at every segment start and stays constant for the segments already in reproduction. The 'bend' box allows altering the reproduction speed while one or more segments are playing. Some other parameters present in the GUI are not specified here.

### Sub-segmenting a loop within the systems time grid:

The sub-segmentation is, without doubt, the most complicated part of the LSM. It consists of three parts, the 'sub-segmentation' section, the 'mangle' player, and the 'trap-loop' player:

The 'sub-segmentation' section to the upper left sets the general parameters for this section. The 'take-seg-nr' and 'rand-seg-nr' toggles to the right are for deciding whether to take the same segment for sub-segmentation than for segmentation or to take a random segment. To the upper left, the reproduction speed and volume are set via the 'ss-speed' and the 'v-mani' number boxes.

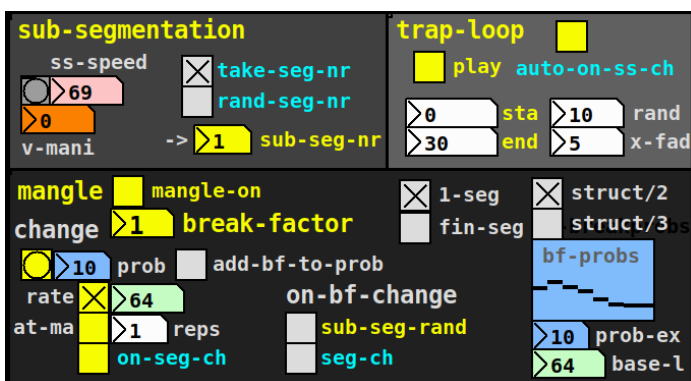


Fig. 67 The 'sub-segmentation' and 'mangle' section.

The 'mangle' section is about resetting the reproduction of the segments to their starting points in shorter periods of time as shown in figure 58. This mechanism is especially interesting for the production of staccato-like rhythmic elements that fit into the time grid of an improvisation. For this purpose, at the lower right in the 'base-l' number box a basic length is defined in pulses. This basic length is then divided by the 'break-factor' into smaller parts. The 'break-factor' specifies the fraction of the reproduced segments and can take the values 1 (no fraction) to 8 (high fraction i.e. short repetitions)<sup>85</sup>. It can be set via the yellow 'break-factor' number-box or changed automatically

85 The break-factors divide the base-length ('base-l') by two successively. So 'break-factor' 1 means to play the whole base-length before resetting to the starting point, 'break-factor' 2 means play half of it and reset, 3 means play a quarter of it, 4 means an eighth, and so on. This dividing by two is exactly what happens when the 'struct/2' toggle is checked. The 'struct/3' toggle divides the base-length by doubles of three successively and produces triplet like rhythmic structures.

depending on the rate and probabilities specified in the blue 'bf-probs' array to the lower right. The break-factor change ('bf-change' in the GUI) is an important factor in the production of rhythmic patterns because it marks the points in time when changes in the mangle behavior (and therefore in the rhythmic appearance) are happening. The 'bf-probs' array at the lower right consists of eight horizontal positions, which represent the 'break-factor' from 1 to 8, and the corresponding probabilities of choosing the respective break-factor. Some further parameters present in the 'mangle' GUI are not specified here.

**The 'trap-loop' section** is experimental and included in the GUI of the LSM at the actual stage of development. The idea is to be able to play long sounds based on the identified segments. The 'trap-loop' mode plays the attack (i.e. the first bit) of the current sub-segment (i.e. the segment assigned to the sub-segment section) and gets then 'trapped' repeating a small section of the segment which can be specified in milliseconds in the yellow number boxes 'sta' for start and 'end'.

**The volume control and loop section** is situated at the upper right of the GUI. It is located after the different play-modes and controls the sum of the audio signals produced by the LSM. The loop function records the current audio production of the LSM according to the specified loop length and plays it in a loop (quantized within the time-grid).

The LSM can produce interesting results with distinct sound material as a base for the generative restructuring carried out by the software. Practical examples<sup>86</sup> include drum loops, samples of recorded music, Foley sounds, speech, field-recordings, etc.

---

86 Practical examples using the LSM can be found on: [werkstsatt.hotglue.me/sound](http://werkstsatt.hotglue.me/sound) (Jan 2021).

### 3 Conclusion and Outlook

In this thesis, I have described a system for generative sequencing in the field of sound design and documented its implementation in the software Puredata. The main goal was to document and conceptualize a creative process that I have been working on in the past years when designing generative sound installations. Therefore, I have first taken a short excursion into the theoretical background that the concept of generating structures through automated or semi-automated processes presupposes (introduction). Based on that, I have drawn a model of the sequencing system that I use and develop (main part). I sketched its important functional components and their interaction to form a sound-emitting production system. In conjunction with the practical applications shown<sup>87</sup>, I have hopefully been able to provide some insight into the following topics:

- The use of generative structuring methods operating on a symbolic level.
- The combination of symbolic productions with a timing mechanism to create temporal sequences using some computer software.
- The mapping of timed sequences to sound parameters with the help of a mapping strategy that transfers them into audible content.
- How to design a functional generative sequencing system that allows both human and machine improvisation and is open to further implementations.

Extending a practical work with a theoretical description and conceptualization has proven to be a way that also benefits the practical work. Designing a conceptual representation of a complex process enables a wider perspective on its possibilities and limitations. Various aspects and implications have been brought to light that often remain in the dark in a purely practical approach. The most important ones are:

- Creating a framework that allows the creative project to be perceived in its entire structure and dimension in order to gain more overview.
- Differentiating a complex practical process into its essential components allows these components to be considered separately in terms of expanding their capabilities and increasing their efficiency.
- Relating one's own practical work to a theoretical discourse provides the opportunity of reflecting and critically questioning one's own approach within a larger historical process.

---

<sup>87</sup> See my webpage [werkstatt.hotglue.me/sound](http://werkstatt.hotglue.me/sound) (Jan 2021) for documentation of the practical applications in the form of audio examples.

For my further work in the field of sound design, the exploration of creative algorithmic processes will remain an important aspect. The use of generative processes in the creation of sound installations and performances is a broad field that I have only started to begin to investigate with this work. With the creation and discussion of an adaptable and expandable generative sequencing system, I formed the base for further works in this field. The new areas of creation and research that have emerged from my work so far are the following:

- The production of more musical content using the methods developed in the course of this work – especially in combination with live sound input.
- The further implementation of generative methods and algorithms into the generative system in use.
- The further implementation of mapping strategies in the production of sound from symbolic sequences.
- The transfer of generative methods and processes to other areas of art and design (sculpture, visual design, landscape architecture, etc.).
- The finding of a more general concept of generative process design that uses generative mechanisms in a broader sense and does not only consider computer algorithms.

## References

- (Arar/Kapur 2013) Arar, Raphael / Kapur, Ajay: *A History of Sequencers: Interfaces for Organizing Pattern Based Music*, in: Proceedings of the Sound and Music Computing Conference 2013, Stockholm, Sweden, pp. 383-388.  
<https://zenodo.org/record/850188> (Jan 2021).
- (Bod 2001) Bod, Rens: *Memory-Based Models of Melodic Analysis: Challenging the Gestalt Principles*, in: Journal of New Music Research, 2001, vol 30.  
<https://www.phil.uu.nl/ozsl/articles/Bod03.pdf> (Jan 2021).
- (Bod 2002) Bod, Rens: *A Unified Model of Structural Organization in Language and Music*, in: Journal of Artificial Intelligence Research 17 (2002) 289-308.
- (Boden 2009) Boden, Margaret / Edmonds, Ernest (2009): *What is Generative Art?*, in: Digital Creativity. 20 (1/2): 21-46.
- (Bowers 1972) Bowers, Q. David: *Encyclopedia of Automatic Musical Instruments*, Maryland, Vestal Press, 1972.
- (Butler 2006) Butler, Mark: *Unlocking the Groove: Rhythm, Meter, and Musical Design in Electronic Dance Music*, Bloomington, Indiana University Press, 2006.
- (Cambouropoulos 1998) Cambouropoulos, Emilios: *Towards a General Computational Theory of Musical Structure*, 1998. [http://users.auth.gr/emilios/papers/phd\\_cambouropoulos\\_1998.pdf](http://users.auth.gr/emilios/papers/phd_cambouropoulos_1998.pdf) (Jan 2021).
- (Cambouropoulos 2008) Cambouropoulos, Emilios: *Voice and stream: perceptual and computational modeling of voice separation*, University of Thessaloniki, 2008.  
<http://users.auth.gr/emilios/papers/MP2008.pdf> (Jan 2021).
- (Cambouropoulos 2009) Cambouropoulos, Emilios: *How similar is similar?* University of Thessaloniki, 2009.  
[http://users.auth.gr/emilios/papers/MS-Discussion\\_Forum\\_4B-Cambouropoulos\\_final.pdf](http://users.auth.gr/emilios/papers/MS-Discussion_Forum_4B-Cambouropoulos_final.pdf) (Jan 2021).
- (Cambouropoulos 2010) Cambouropoulos, Emilios: *The Musical Surface: Challenging Basic Assumptions*, in: Musicae Scientiae 2010, p. 131-147.  
<http://users.auth.gr/emilios/papers/cambouropoulos-MS2010.pdf> (Jan 2021).
- (Chomsky 1964) Chomsky, Noam: *Current Issues in Linguistic Theory*, Mouton, 1964.
- (Dąbrowska 2015) Dąbrowska, Ewa: *What exactly is Universal Grammar, and has anyone seen it?* in: Frontiers in Psychology, Jun 2015.  
<https://www.frontiersin.org/articles/10.3389/fpsyg.2015.00852/full> (Jan 2021).



- (Dale 2008) Dale, Michael: *What is Comprovisation*, Mills College, 2008.  
<http://www.michaeldmusic.com/thesisFINAL.pdf> (Jan 2021).
- (Davis 2002) Davies, Hugh: *Sequencer*, in: *The New Grove Dictionary of Music and Musicians*, London, Grove 2002, p. 108.
- (Dorin 2000) Dorin, Alan: *Boolean Networks for the Generation of Rhythmic Structure*, in: *Proceedings ACMC 2000*, Brown & Wilding (eds), Australian Computer Music Conference, July 2000, p 38-45.
- (Eco 2002) Eco, Umberto: *Einführung in die Semiotik*, Fink, 2002 (1972).
- (Ekelund 2013) Ekelund, Thomas; Mårtensson, Jon: (2013). *OCTATRACK DPS-1 USER'S MANUAL for operating system version 1.25*. Elektron Music Machines, Gothenburg, Sweden, 2013.
- (Essl 1996) Essl, Karlheinz: *Strukturgeneratoren: algorithmische Komposition in Echtzeit*, Graz, Institut Für Elektronische Musik (IEM), 1996.
- (Farnell 2010) Farnell, Andy: *Designing Sound*, MIT Press, 2010.
- (Fowler 1967) Fowler, Charles B.: *The Museum of Music: A History of Mechanical Instruments*, in: *Music Educators Journal*, October 1967, 45-49.
- (Henckmann/Lotter 2004) Henckmann, Wolfhart; Lotter, Konrad: *Lexikon der Ästhetik*, München, Verlag C.H. Beck, 2004.
- (Hillerson 2014) Hillerson, Tony: *Programming Sound with Pure Data: Make Your Apps Come Alive with Dynamic Audio*, Pragmatic Book Shelf, 2014.
- (Lerdahl 2009) Lerdahl, Fred: *Genesis and Architecture of the GTTM Project*, in: *Music Perception, an Interdisciplinary Journal*, volume 26, 2009, p. 185-301.
- (Lerdahl/Jackendoff 1983) Lerdahl, Fred / Jackendoff, Ray: *A Generative Theory of Tonal Music*, MIT Press, 1983.
- (Lindemayer/Prusinkiewicz 2004) Lindemayer, Aristid / Prusinkiewicz, Przemyslav: *The Algorithmic Beauty of Plants*, Springer, 2004.
- (Merleau-Ponty 1963) Merleau-Ponty, Maurice: *The Structure of Behavior*, Boston, 1963 (1945).
- (Münker/Roesler 2000) Münker, Stefan / Roesler, Alexander: *Poststrukturalismus*, J. B. Metzler, 2000.
- (Nierhaus 2008) Nierhaus, Gerhard: *Algorithmic Composition*, Springer, 2008.
- (Nierhaus 2015) Nierhaus, Gerhard: *Patterns of Intuition*, Springer, 2015.

(Nierhaus 2021) Nierhaus, Gerhard: *On Composers and Computers*. To be published in *Oxford Handbook of the Creative Process in Music*, ed. Nicolas Donin. Oxford University Press, 2021.

<https://gerhardnierhaus.com/on-composers-and-computers> (Jan 2021).

(Nierhaus 2021/2) Nierhaus, Gerhard: *Constraints*, foreword in: *Composing with Constraints* by Jorge Variego, to be published by Oxford University Press in 2021.

<https://gerhardnierhaus.com/constraints> (Jan 2021).

(Puckette 2007) Puckette, Miller: *Theory and Techniques of Electronic Music*, World Scientific Publishing CO, 2007.

<http://msp.ucsd.edu/techniques/latest/book.pdf> (Jan 2021).

(Saussure 1995) Saussure, Ferdinand de: *Cours de linguistique générale*, Bibliothèque scientifique Payot, 1995 (1967).

(Schaeffer 1966) Schaeffer, Pierre: *Traité des objets musicaux*, Seuil, Paris, 1966.

(Wertheimer 1923) Wertheimer, Max: *Untersuchungen zur Lehre von der Gestalt*, in: *Psychologische Forschung* 4, 301–350, 1923.

[http://gestalttheory.net/download/Wertheimer1923\\_Lehre\\_von\\_der\\_Gestalt.pdf](http://gestalttheory.net/download/Wertheimer1923_Lehre_von_der_Gestalt.pdf) (Jan 2021).

(Zotter/Frank 2019) Zotter, Fanz / Frank, Matthias: *Ambisonics, A Practical 3D Audio Theory for Recording, Studio Production, Sound Reinforcement, and Virtual Reality*, Springer, 2019.