

# Stochastic processes and probability theory in music

Monika Jadwiga Galla

Master's by Research  
University of York  
Music  
December 2015

## **Abstract**

This dissertation examines the connections between music and mathematics with particular reference to Markov chains and generative grammars. The main purpose of this study is to investigate how mathematical concepts can help to control, create and analyse music material. The core part of this study is software that allows one to compose music with Markov Chains and generative grammars. The study will explore the on-going influence of such tools on composers and their relationship to musical sources and inspirations.

An in-depth analysis of existing literature, music material and composition tools was conducted. Using comparative case studies, this research explored the significant role of mathematics in music in the twentieth and twenty-first centuries. The evolving role of stochastic concepts in music was presented. The next step was to develop a useful tool that would allow composers to apply Markov chains and generative grammars in their compositions. The web application that resulted was called Stochastic Composer. To evaluate this application five composers were invited to test it. The results include over one hundred samples of music material that were later analysed and used to improve the software.

This dissertation offers insight into applications of various mathematical concepts in music. The Stochastic Composer software, available online, proved to be a useful tool in a compositional process.

## Table of Contents

<b>Abstract .....</b>	<b>2</b>
<b>Table of Contents.....</b>	<b>3</b>
<b>List of Figures .....</b>	<b>5</b>
<b>List of Tables.....</b>	<b>8</b>
<b>List of accompanying material .....</b>	<b>9</b>
<b>Acknowledgements [ENG].....</b>	<b>10</b>
<b>Acknowledgements [PL] – Podziękowania .....</b>	<b>11</b>
<b>Author's declaration.....</b>	<b>12</b>
<b>Overview.....</b>	<b>13</b>
<b>1. Markov Chains .....</b>	<b>15</b>
<b>1.1. Introduction.....</b>	<b>15</b>
<b>1.2. Basic principles of Markov chains.....</b>	<b>16</b>
1.2.1. Properties of Markov chains.....	18
1.2.2. Hidden Markov Models.....	23
1.2.3. Example.....	24
1.2.4. Markov Decision Process (MDP).....	26
<b>1.3. Music.....</b>	<b>27</b>
1.3.1. Possible applications .....	27
1.3.2. Prediction.....	27
1.3.3. Composition .....	33
<b>1.4. Summary .....</b>	<b>42</b>
<b>2. Generative grammars .....</b>	<b>44</b>
<b>2.1. Introduction.....</b>	<b>44</b>
<b>2.2. Basic concept of generative grammars .....</b>	<b>45</b>
2.2.1. Chomsky Hierarchy.....	48
<b>2.3. Grammars in music .....</b>	<b>50</b>
2.3.1. Overview .....	50
2.3.2. Composing using generative grammars .....	51
2.3.3. Generative Grammar Definition Language .....	51
2.3.4. Generative theory of tonal music .....	53
2.3.5. Web Grammars.....	57

2.3.6.	Stochastic grammars.....	61
2.3.7.	Problem of termination.....	64
<b>2.4.</b>	<b>Summary .....</b>	<b>64</b>
<b>3.</b>	<b>Stochastic Composer .....</b>	<b>65</b>
<b>3.1.</b>	<b>Introduction.....</b>	<b>65</b>
<b>3.2.</b>	<b>Software design description .....</b>	<b>68</b>
3.2.1.	Web Application Concept .....	68
3.2.2.	Server side .....	69
3.2.3.	Database .....	71
3.2.4.	User Interface .....	72
3.2.5.	Deployment .....	73
3.2.6.	MusicXML.....	73
<b>3.3.</b>	<b>Algorithm Implementation .....</b>	<b>76</b>
3.3.1.	Music Notation .....	76
Markov Chains .....		77
3.3.2.	Generative Grammars.....	81
3.3.3.	Navigation .....	84
3.3.4.	Markov Chains .....	87
3.3.5.	Generative Grammars.....	90
<b>3.4.</b>	<b>Results.....</b>	<b>95</b>
3.4.1.	Markov Chains .....	95
3.4.2.	Generative Grammars.....	107
<b>3.5.</b>	<b>Summary .....</b>	<b>121</b>
	<b>Final remarks.....</b>	<b>122</b>
	<b>Bibliography.....</b>	<b>123</b>

## List of Figures

Figure 1: Simple melody with missing note .....	16
Figure 2: Completed melody from Figure 1 .....	16
Figure 3: Soft Kitty: melody.....	17
Figure 4: Soft Kitty Transition Matrix .....	18
Figure 5: Soft Kitty - transition graph .....	18
Figure 6: Closed and open classes, absorbing event.....	19
Figure 7: A chain with only transient classes, and hence not a Markov chain .....	20
Figure 8: A null recurrent Markov chain ( $n \rightarrow \infty$ ) .....	21
Figure 9: A positive recurrent Markov chain .....	21
Figure 10: A transient Markov chain.....	21
Figure 11: Markov chain with period 3.....	22
Figure 12: Markov chain with period 2.....	23
Figure 13: A Markov chain with period 1, an aperiodic Markov chain .....	23
Figure 14: HMM example: a pianist as a neighbour .....	25
Figure 15: Results of the experiment (Brooks, Hopkins, Neumann, & Wright, 1957, p. 181).....	28
Figure 16: Constructed Markov chain (Verbeurgt, Dinolfo, & Fayer, 2004b).....	29
Figure 17: A piece written on the basis of Bach's 'Air' using patterns.....	29
Figure 18: A piece written on the basis of Bach's 'Air' using a typical Markov chain approach.....	30
Figure 19: Harmonization using HMM .....	31
Figure 20: Original harmonization by Bach.....	32
Figure 21: Harmony generates with MDP.....	33
Figure 22: Matrix for chromatic degrees in Demonstration 4 by C. Ames .....	35
Figure 23: Charles Ames, Demonstration 4 .....	36
Figure 24: A toroidal space (Virtual Math Museum).....	38
Figure 25: Chord generation (McAlpine, Hoggar, & Miranda, 1999, p. 26) .....	38
Figure 26: Sample state space (I) (Jones K. , 1980, p. 98).....	40
Figure 27: Sample state space (II), (Jones K. , 1980, p. 112).....	40
Figure 28: Sample state space for clarinet (Jones K. , 1980, p. 121) .....	41
Figure 29: Result of Grammar 1 presented as parse .....	48
Figure 30: Chomsky Hierarchy .....	49
Figure 31: Analysis of bars 9-16 from Andante from Haydn's Symphony no 94.....	55

Figure 32: Time-span reduction.....	56
Figure 33: Time-span reduction.....	56
Figure 34: Music representation of the outcome of Grammar 4 .....	60
Figure 35: Stochastic Composer - Software Structure.....	69
Figure 36: Stochastic Composer - Database Structure .....	72
Figure 37: MusicXML Example - Score.....	74
Figure 38: Music XML - Simple Example .....	75
Figure 39: Stochastic Composer - Music Notation Classes .....	76
Figure 40 - Stochastic Composer - Markov Chain Generation Algorithm .....	79
Figure 41 - Stochastic Composer - Markov Chain Prediction/Simulation Algorithm .....	80
Figure 42 - Stochastic Composer - Grammar Outcome Generation Algorithm .....	83
Figure 43 - Stochastic Composer - Gaussian Kernel Example.....	83
Figure 44: Welcome screen in Stochastic Composer .....	84
Figure 45: Learning Centre options.....	85
Figure 46: Compose section .....	85
Figure 47: My Compositions .....	86
Figure 48: Composition Details .....	86
Figure 49: Details of the composition.....	87
Figure 50: Melody created in Stochastic Composer .....	87
Figure 51: Markov chain details .....	88
Figure 52: Markov chain generated output.....	88
Figure 53: Saving output in a database .....	89
Figure 54: My compositions section .....	89
Figure 55: Details of Markov chain composition.....	89
Figure 56: Start Symbol.....	90
Figure 57: Adding new production rule.....	91
Figure 58: Successful validation.....	91
Figure 59: Generated Output .....	92
Figure 60: Stochastic Production Rule.....	92
Figure 61: Probability of the rules are not equal to 1 .....	93
Figure 62: Production rules in web grammars.....	93
Figure 63: Combined stochastic and web production rules .....	94
Figure 64: Stochastic Composer - Example MC1.....	95

Figure 65: Stochastic Composer - Example MC1 - loop.....	96
Figure 66: Stochastic Composer -Example - MC2.....	96
Figure 67: Stochastic Composer -Example - MC3.....	97
Figure 68: Stochastic Composer - Example - MC 4.....	100
Figure 69: Stochastic Composer - Example - MC 5 Input.....	102
Figure 70: Stochastic Composer -Example - MC5a.....	102
Figure 71: Stochastic Composer -Example - MC5b.....	103
Figure 72: Stochastic Composer -Example - MC5c.....	105
Figure 73: Stochastic Composer - GSC1 - Sample 1 .....	108
Figure 74: Stochastic Composer - GSC1 - Sample 2 .....	108
Figure 75: Stochastic Composer - GSC1 - Sample 3 .....	109
Figure 76: Stochastic Composer - GSC1 - Sample 4 .....	109
Figure 77: Stochastic Composer - GSC1 - Sample 5 .....	110
Figure 78: Stochastic Composer - GSC2 - Sample 1 .....	112
Figure 79: Stochastic Composer - GSC2 - Sample 2 .....	112
Figure 80: Stochastic Composer - Production Rule Probability Distribution for Note 'A'.....	114
Figure 81: Stochastic Composer - GSC3 - Sample 1 .....	116
Figure 82: Stochastic Composer - GSC3 - Sample 2 .....	116
Figure 83: Stochastic Composer - GSC - Markov chain from generative grammar.....	116

## List of Tables

Table 1: Comparison of recurrent and transient states .....	20
Table 2: Possible generations of Grammar 1 .....	46
Table 3: Possible generations by Grammar 2.....	47
Table 4: Sample LHS transition matrix in GGDL .....	52
Table 5: Assigning probabilities to Grammar 5.....	62
Table 6: Production rules for a Soft Kitty Grammar .....	63
Table 7: Production rules for Grammar 8 .....	64
Table 8: Problem of termination .....	64
Table 9: Stochastic Composer - Example MC1 .....	95
Table 10: Stochastic Composer - Example - MC2 .....	96
Table 11: Stochastic Composer -Example - MC3 .....	97
Table 12: Stochastic Composer - Example - MC 4 - Transition Diagram.....	101
Table 13: Stochastic Composer - Example - MC5a - Transition diagram.....	102
Table 14: Stochastic Composer -Example - MC5b - Transition diagram.....	103
Table 15: Stochastic Composer -Example - MC5c - Transition diagram.....	105
Table 16: Stochastic Composer - GSC2 - Sample 1 - Generations .....	112
Table 17: Stochastic Composer - GSC2 - Sample 2 - Generations .....	112
Table 18: Stochastic Composer - GSC3 - Markov Chain from generative grammar - Transition diagram .....	117



## List of accompanying material

A cloud based composition software is available at:

<http://stochasticcomposition.azurewebsites.net/>

## **Acknowledgements [ENG]**

Firstly, I would like to express my sincere gratitude to my supervisor William Brooks for his continuous support through my Master's study and related research—for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not imagine having a better supervisor and mentor for my MRes study.

My sincere thanks also goes to my beloved husband, Daniel Pecynski, for his invaluable help in creating the Stochastic Composer programme, the hours spent explaining programming and software terminology, stimulating discussions and motivation. While the responsibility for the final product is mine alone, without his precious support it would not have been possible to conduct this research.

Special thanks also go to all my composer friends for participating in the testing of Stochastic Composer, and especially for their enthusiasm. And thanks also go to all the participants in my survey, for their enthusiastic involvement in my research as well as insightful suggestions and comments.

Last but not least, I would like to thank my parents for supporting me spiritually throughout writing this thesis and my life in general. Thank you for your support and unconditional love. Even though we are thousands of miles apart, you were always there whenever I needed you. You can take all the credit for what I have achieved thus far and what I will achieve in the future.

## Acknowledgements [PL] – Podziękowania

Przede wszystkim chciałabym wyrazić szczerą wdzięczność mojemu Profesorowi Williamowi Brooks za jego nieustanne wsparcie podczas studiów i badań, za jego cierpliwość, motywacje i olbrzymia wiedze. Jego wskazówki niezwykle pomogły mi w trakcie pisania tej pracy. Nie mogłam sobie wyobrazić lepszego promotora i mentora dla moich badań.

Ogromne podziękowania należą się również mojemu mężowi, Danielowi Pecyńskiemu, za nieocenioną pomoc podczas tworzenia programu Stochastic Composer, za godziny poświęcone tłumaczeniu mi programowania i procesu tworzenia aplikacji, za niezliczone dyskusje i zaangażowanie. Bez jego cennego wsparcia ta praca nie byłaby możliwa.

Specjalne podziękowania kieruje również do moich przyjaciół kompozytorów za udział w testowaniu Stochastic Composer, za ich entuzjazm, a także wnikliwe sugestie i uwagi, a także wszystkim, którzy wzięli udział w ankiecie przeze mnie organizowanej za ich zaangażowanie i ciekawe komentarze, które pomogły mi w badaniach.

Na koniec chciałabym podziękować moim rodzicom i babciom za wspieranie mnie duchowo przez całe moje życie, a szczególnie podczas pisania tej pracy. Dziękuję Wam za wsparcie i bezwarunkową miłość. Mimo że jesteście tysiące mil, zawsze byliście obecni gdy Was potrzebowałam. To Wam zasługuje wszystko, co do tej pory osiągnęłam i co osiągnę w przyszłości.

## **Author's declaration**

I hereby declare that I am the sole author of this thesis, except where referenced.

Parts of this thesis were previously published in:

*Galla-Pecynska, Monika. Markov Chain Application in Creating and Understanding Music. Proceedings of the 1st Sound Ambiguity Conference. 2014.*

The interpretations put forth are based on my reading and understanding of the original texts and they are not published anywhere in the form of books, monographs or articles. The other books, articles and websites that I have made use of are acknowledged at the respective places in the text. This work has not been submitted for another award at this, or any other institution.

## Overview

This thesis investigates and analyses stochastic processes and probability-theory influences on compositional processes and also explores new methods and tools of analysis developed through mathematic research.

The choice of the subject was dictated by the author's interest in both musical and mathematical concepts. The author studied at the University of Technology in Wroclaw (Poland), where she obtained a Bachelor's and a Master's degree in Management. During her studies she investigated mainly game theory and database processing. At the same time, in Poland, she obtained a Bachelor's degree in Composition and Music Theory at Wroclaw Academy of Music. Her final thesis was entitled *Game Theory in Iannis Xenakis' Works*. This thesis constitutes a natural development of the author's research interests.

The work consists of two parts. The first contains two theoretical chapters: Markov chains and Generative grammars. These contain comprehensive literature overviews with relevant examples for each of the subjects discussed. In this part the author focuses mainly on the theoretical bases of general stochastic processes, Markov chains, and generative grammars. The study examines the influence of these both on compositional processes and on musical works themselves. In addition to theory, certain experiments are illustrated using appropriate musical examples. This part also offers an overview of technological developments created by and for composers as well as music theoreticians, and it explores their possible applications.

The second part is focused mainly on creating and developing already existing tools, which use stochastic processes, for music purposes. This part consist of chapter 3 and the associated Stochastic Composer software, available at <http://stochasticcomposition.azurewebsites.net>. The objective in this part was to create a complete suite of tools that allows the composition and analysis of music using Markov chains and generative grammars. This is how the web application Stochastic Composer was conceived. The application allows composers to create and design their own generative grammars and to connect these with other stochastic processes. Stochastic Composer allows a user not only to compose but also to consider music prediction based on Markov chains. The user needs to provide a necessary amount of

musical material; then the program will analyse the sample and calculate the Markov chain of the required order. The analysis can be then used to generate music material that will conform to a greater or lesser degree to the music of the composer or style in question. In particular, the Markov and generative grammar tools can be used in tandem: a composer can generate material from a generative grammar and then develop it independently by the use of Markov analysis and composition. This approach to composition and analysis concludes with examples of possible program outputs.

# 1. Markov Chains

## 1.1. Introduction

On 23rd January 1913 Andrei Andreyevich Markov presented his analysis of the first 20,000 letters of Pushkin's *Eugene Onegin* in an address to the Imperial Academy of Sciences in St. Petersburg. While his findings did not explain the beauty or story behind Pushkin's poem, he started a new chapter in probability theory that is now known as Markov chains. In his publication Markov wrote: 'Let us finish the article and the whole book with a good example of dependent trials, which approximately can be regarded as a simple chain' (Markov, 1913, cited in Basharin, Langville, & Naumov, 2004, p.16). The example contained a study of a sequence of 20,000 letters in A. S. Pushkin's poem *Eugeny Onegin* and revealed that the stationary vowel probability equals  $p = 0.432$ , the probability of a vowel following a vowel is  $p_1 = 0.128$ , and the probability of a vowel following a consonant is  $p_2 = 0.663$ . These simple calculations, with great attention to detail, are now recognised as the first application of Markov chains.

The 1913 publication also contained results of Markov's other tests: he studied a sequence of 100,000 letters in S. T. Aksakov's novel *The Childhood of Bagrov, the Grandson*. In this case he achieved the following results: vowel probability  $p = 0.449$ , probability of a vowel following a vowel  $p_1 = 0.552$ , and probability of a vowel following a consonant  $p_2 = 0.365$  (Basharin, Langville, & Naumov, 2004). Over a hundred years later Markov Chains have been applied successfully to sociology (Kemeny, Snell, & Knapp, 2011); to physics, to simulate the collective behaviour of systems created of many interacting particles (Hayes, 2013); to biology, for the purpose of analysing DNA sequences (Schbath, Prum, & de Turckheim, 1995; Almagor, 1983); or even to analyze baseball (Pankin). Significant work has been done in the field of information theory, with C. Shannon's paper 'A Mathematical Theory of communication' at the forefront (Shannon, 1948). After Lejaren Hiller's composition of the *Illiac Suite* (1956) Markov chains have also become a remarkable part of stochastic composition and have occupied an important place in twentieth-century understandings of music.

## 1.2. Basic principles of Markov chains

As Markov showed in 1913, it is possible to predict whether the next speech sound will be a vowel or a consonant. Later studies confirmed his results, and they show that in some cases it is possible to predict precisely even the next letter. For example, in the English language the letter following Q will always be U. Similarly, for a given three-letter word that starts with the letters WA, one can predict the whole word: it can be war, was, way or wag. Such prediction is not limited only to the English language; an interesting study of eleven languages was conducted by E. B. Newman (1951).

Similar rules seem to apply to music. Even a non-musician (who may, however, be unaware of this) listening to a tonal piece will anticipate a form of tonic as a resolution after a dominant. In a simple music example (Figure 1) one can expect, among several different possibilities, that the next note would be f<sup>1</sup>:



Figure 1: Simple melody with missing note



Figure 2: Completed melody from Figure 1

The answer given in Figure 2 is only one among many possibilities; however, one can predict with assurance that the next note will not be c<sup>#2</sup> or e b<sup>1</sup>.

On the basis of these linguistic and music examples it is possible to formulate an intuitive definition for a Markov chain: when certain events occur, one can predict that a specific event is more or less likely to follow. The formal definition is presented below:

*Let  $X_0, X_1, \dots$  be a sequence of random variables with possible outcomes  $X_t = x_t \in S$ . Then the sequence is called a Markov chain, if*

*M1. The state space  $S$  is finite or countable;*

*M2. For any  $t \in \mathbb{N}$ ,*

$$P(X_{t+1} = j | X_0 = i_0, X_1 = i_1, \dots, X_t = i_t) = P(X_{t+1} = j | X_t = i_t)$$

(Beran, 2004, p. 176).

The elements of  $S$  are called states and the equation is usually described as the Markov property of the Markov chain  $X_t, t \in \mathbb{N}$ .



The change from one event to the next is commonly referred as a transition, and it therefore must follow the rule

$$p_{i,j} \geq 0, \sum_{j=1}^m p_{i,j} = 1$$

for each  $i= 1,2,\dots, m$ .

The transition probabilities can be presented in a transition matrix, which can be presented in two ways:

- A row stochastic matrix (right stochastic matrix), where each row sums to 1.
- A column stochastic matrix (left stochastic matrix), where each column sums to 1.

A special type of matrix is a double matrix, where both rows and columns sum to 1. The two types of matrices differ only in their visual representation, and so for the purposes of this thesis only row (right) stochastic matrices will be used.

If the transition probabilities are the same at every step (so they are independent of the time step  $t$ ) then the chain is said to be homogeneous.

$$p_{i,j} = P(X_t = j | X_{t-1} = i) = P(X_1 = j | X_0 = i)$$

Another possible representation of Markov Chains is in a transition graph, where each transition is represented as an arrow.

### **An example**

Consider the kindergarten song “Soft Kitty,”<sup>1</sup> shown in Figure 3.



Figure 3: Soft Kitty: melody

The state space  $S$  contains 5 states (c, d, e, f, g), so it is finite and countable. A quick analysis of the event predecessors produced a transition matrix, shown in figure 4, and the transition graph shown in Figure 5.

---

<sup>1</sup> This is the Polish melody *Wlazl kotek na plotek*, written in the nineteenth century by S.Moniuszko as a three-voice canon (with slightly faster tempo and rhythm) and first published in O. Kolberg’s book *Dziela wszystkie* in 1871 (Kolberg 1961).

	C	D	E	F	G
C		1			
D	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{6}$		
E			$\frac{2}{5}$	$\frac{3}{5}$	
F		$\frac{1}{2}$		$\frac{1}{4}$	$\frac{1}{4}$
G			$\frac{1}{2}$		$\frac{1}{2}$

Figure 4: Soft Kitty Transition Matrix

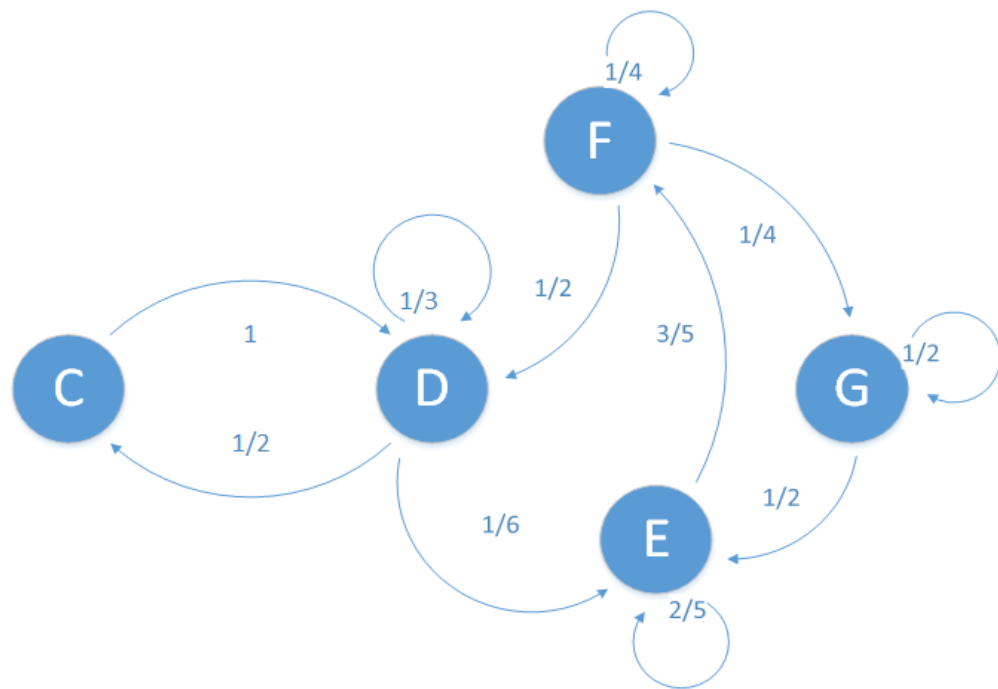


Figure 5: Soft Kitty - transition graph

The number of past events that are taken into consideration at each stage is known as the order of chain. In the ‘Soft Kitty’ example, only an event’s immediate predecessor was taken into consideration when calculating the transition matrix; thus this is a first-order Markov chain.

### 1.2.1. Properties of Markov chains

If there is a positive transition from state  $a$  to state  $b$  then  $b$  is said to be accessible from  $a$  (notated as  $a \rightarrow b$ ). In the ‘Soft Kitty’ example, state (pitch)  $e$  is accessible from state (pitch)  $d$ . This relation is transitive; i. e., if  $d \rightarrow e$  and  $e \rightarrow f$ , then  $d \rightarrow f$ . When two states are accessible from each other they are said to communicate (notated as  $\leftrightarrow$ ). In the previous example states (notes)  $c$  and  $d$  communicate with each other ( $c \leftrightarrow d$ ). A less obvious example is the communication between states  $e$  and  $f$ :

$$\text{if } e \rightarrow f, f \rightarrow g \text{ and } g \rightarrow e, \text{ then } f \rightarrow e \Rightarrow e \leftrightarrow f$$

That is, as can be seen on the transition graph,  $f$  is accessible from  $e$ . Because  $g$  is accessible from  $f$  and  $e$  is accessible from  $g$ , then  $e$  is accessible from  $f$ . Because  $e$  and  $f$  are accessible from each other they are said to communicate.

States that communicate are in an equivalence relation, which means that the relation is:

- Reflective:  $c \leftrightarrow c$
- Symmetric:  $d \leftrightarrow c$  if and only if  $c \leftrightarrow d$
- Transitive: if  $c \leftrightarrow d$  and  $d \leftrightarrow a$ , then  $c \leftrightarrow a$ .

It is possible to split each chain into groups of events that communicate only with each other; these are equivalence classes of events. While events from one class cannot communicate with events from another class, they can be accessible from them. It can be easily seen that the ‘Soft Kitty’ example contains only one communication class: all events communicate with each other.

If every state in a class  $C$  has the property

$$i \in C, i \rightarrow j \Rightarrow j \in C$$

then  $C$  is said to be a closed class. If, in a class  $C$ ,  $i \in C$  and  $j \notin C$  with  $i \rightarrow j$  then  $C$  is said to be open. In other words there is a possibility of escape from this class, while there is no escape from a closed class. A state  $i$  is called absorbing if  $\{i\}$  forms a closed class; once the Markov chain enters this state it is impossible to leave it. In Figure 6 the class  $\{a,b,c\}$  is open—there is a possible escape. The class  $\{g,h\}$  is closed, while the event  $\{e\}$  is an absorbing event.

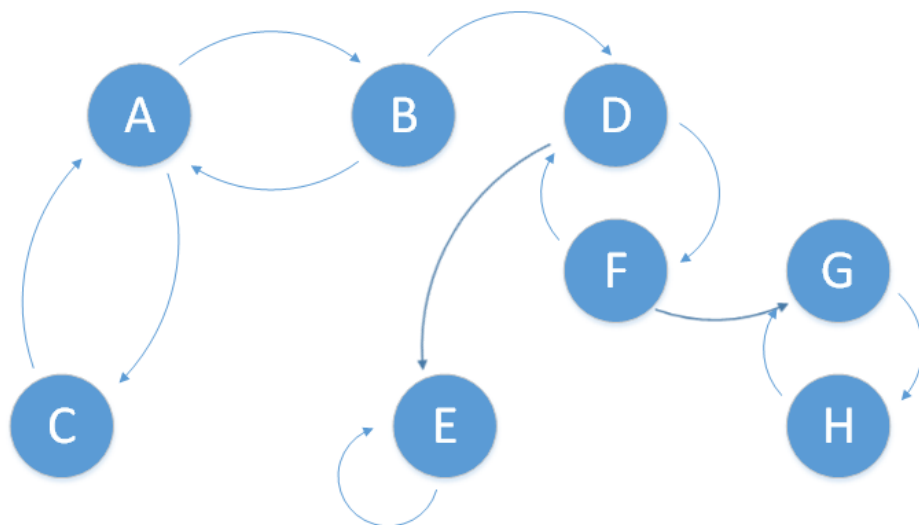


Figure 6: Closed and open classes, absorbing event

Every state is either transient or recurrent. If, after achieving some state  $i$ , the chain will, with probability 1, return to state  $i$ , then the state is said to be recurrent. On the contrary, if when the chain achieves some state  $i$ , there is a positive probability ( $0 < p < 1$ ) that the chain will never return to the state, then the state is said to be transient. All states that communicate (so belonging to one class) with a recurrent state are also recurrent and form a recurrent class. All states that communicate (so belonging to one class) with a transient state are also transient and form a transient class. Every Markov chain must contain at least one recurrent class. Figure 7 illustrates a chain with only transient classes, which is therefore not a Markov chain. A Markov chain with only one recurrent class is said to be irreducible. That means that regardless of the starting state it is possible to reach any other state in a finite number of steps. The ‘Soft Kitty’ example contains one recurrent class, so it is irreducible.

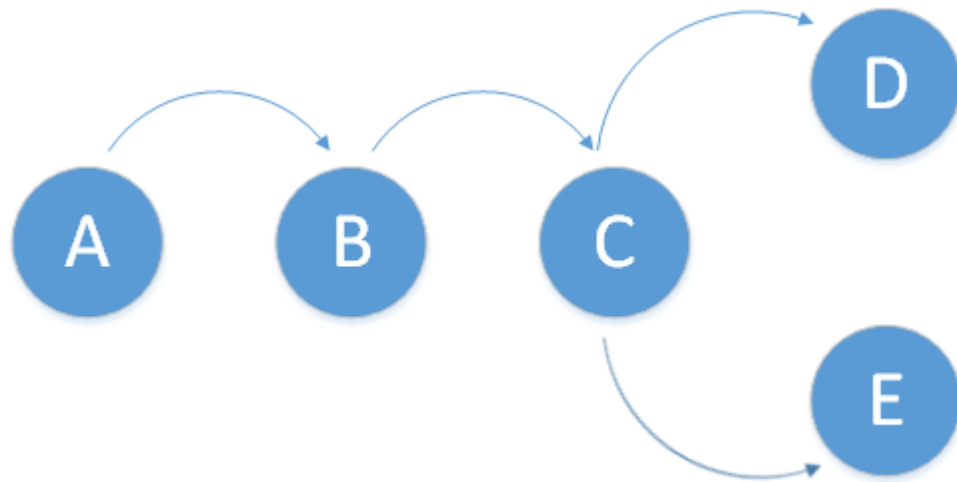


Figure 7: A chain with only transient classes, and hence not a Markov chain

It is possible to distinguish between positive recurrent and null recurrent Markov chains. If the probability of returning to state  $i$  is 1 and the expected recurrent time is finite, then the chain is positive recurrent. If a probability of return to state  $i$  is 1 but the expected recurrent time is infinity, then the chain is said to be a null recurrent. It is very important to distinguish null recurrent states from transient states. Transient states may be never visited again, while null recurrent states will certainly be visited again, but in an infinite amount of time (Table 1).

Table 1: Comparison of recurrent and transient states

Name	Probability of return to state	Time
Positive recurrent	1	$< \infty$
Null recurrent	1	$\infty$
Transient	$< 1$	$< \infty$

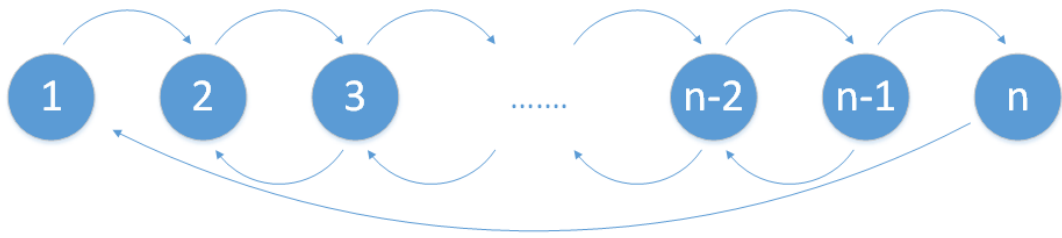


Figure 8: A null recurrent Markov chain ( $n \rightarrow \infty$ )

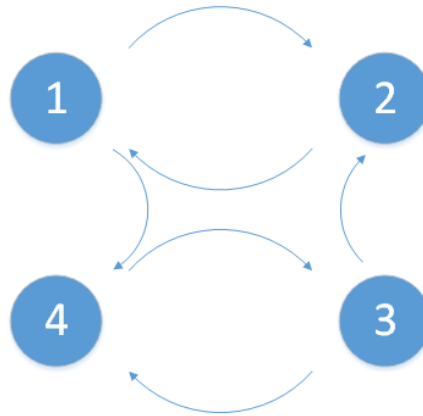


Figure 9: A positive recurrent Markov chain

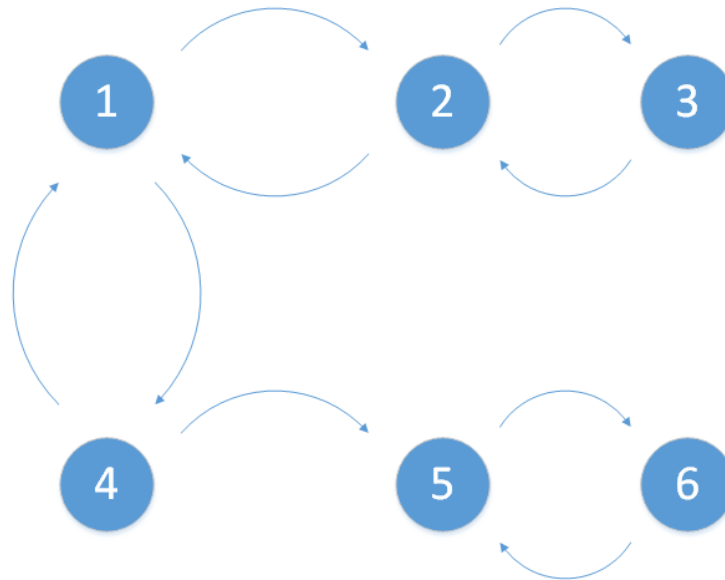


Figure 10: A transient Markov chain

Figure 8 presents a null recurrent Markov chain. The number of states is  $n$ , where  $n \rightarrow \infty$ . Although starting from state 1 it is certain that eventually the chain will return to state 1, this may be achieved only after an infinite length of time. In contrast, the positive recurrent Markov chain presented in Figure 9 will return to its starting state in a finite length of time. Figure 10 presents a transient Markov chain: starting from state 1 it is

not certain that the chain will return to the state 1, since it may get stuck in the closed class {5,6}.

### **Periodicity**

The period of a chain is defined as  $k$ , when  $k = \text{GCD} \{n: P(X_n = i | X_0 = i) > 0\}$ , where GCD states for greatest common divisor. In other words the return to state  $i$  will occur after a multiple of  $k$  steps. When a state can be revisited in irregular numbers of steps—that is, when  $k=1$ —the state is said to be aperiodic. If a state occurs in regular time steps—that is, when  $k>2$ —then the state is said to be periodic, with period  $k$ . A Markov chain is aperiodic if every state is aperiodic. The period of the states in a class is a property of that class; that is, all states that communicate (so they are from the same communicating class) share the same period. It follows that an irreducible Markov chain needs only one aperiodic state to imply that all states are aperiodic. In the example presented in Figure 11 the period is  $k=3$ , while the example presented in Figure 12 has the period  $k=2$ . Figure 13 presents a chain with period  $k=1$ , which means that the chain is aperiodic.

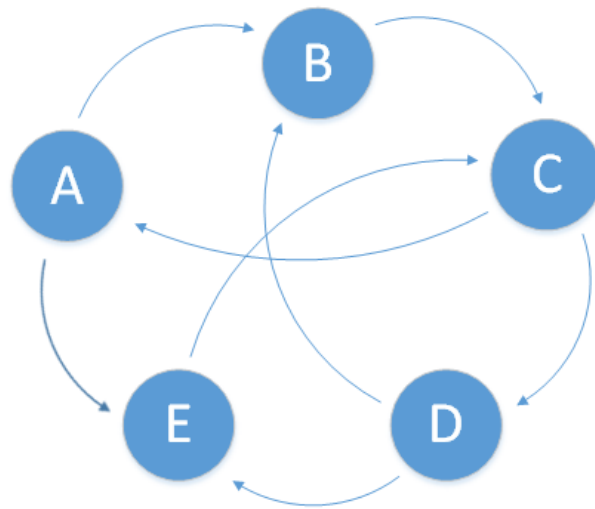


Figure 11: Markov chain with period 3

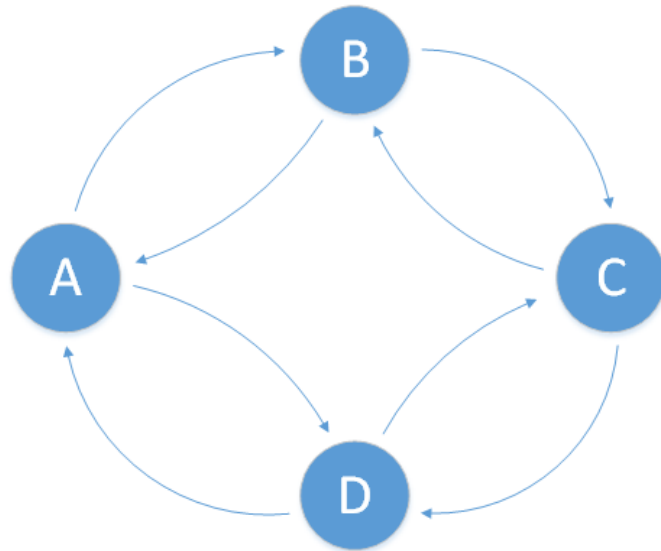


Figure 12: Markov chain with period 2

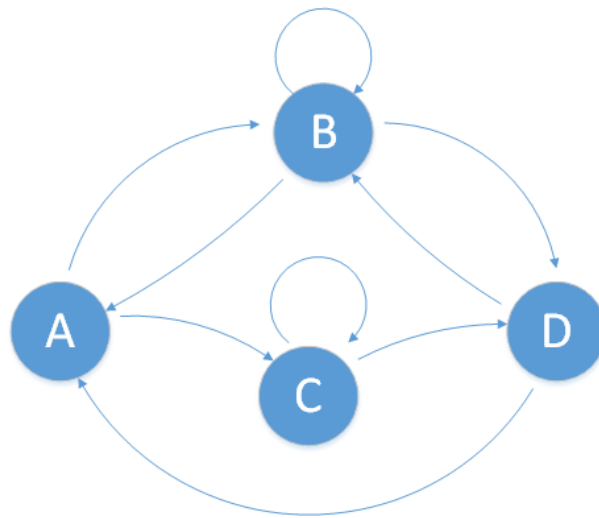


Figure 13: A Markov chain with period 1, an aperiodic Markov chain

Markov chains are described in greater detail in Brzezniak and Zastawniak (1999), Jones and Smith (2001), and Romanovsky (1970).

### 1.2.2. Hidden Markov Models

As described above, Markov chains are often said to be too restrictive and not sufficient to address some problems. An extended version of the Markov concept—Hidden Markov Models (HMM)—seems to offer one solution. In a hidden Markov model, a sequence of  $M$  observations,  $O_1$  to  $O_M$ , is generated by a sequence of hidden  $N$  states,  $S_1$  to  $S_N$ . Hidden states, as the name suggests, cannot be observed directly. Transitions between hidden states are assumed to form a first order Markov chain. The transition probabilities are defined as:  $A = \{a_{ij}\}$ , where  $a_{ij} = P(q_{t+1} = S_j | q_t = S_i)$ , with

the initial state probabilities  $\pi = \{\pi_i\}$ , where  $\pi_i = P(q_1 = S_i)$ . The observation probabilities are defined as  $B = \{b_j(k)\}$ , where  $b_j(k) = P(v_t = O_k | q_t = S_j)$ . A HMM, then, can be represented as  $M=(A, B, \pi)$ . It is important to note that a HMM requires not just one matrix but two: the first, as in the previous model, to determine the transition probabilities, and the second to determine the output probabilities.

### 1.2.3. Example

Suppose that person X has a neighbour who is a pianist. The pianist usually plays piano repertoire that goes with her mood: when she is melancholy she plays Chopin, when she is happy she plays Mozart, when she is sad she plays Tchaikovsky. Person X can then make assumptions about the pianist's mood based on the repertoire that he can hear through the wall. In this case the sounds can be described as *observations*, while the real mood of the artist will be *hidden*. Figure 14 contains a transition graph for this. The initial probabilities  $\pi$  are shown in blue, with solid lines; transition probabilities  $a$  are shown in orange colour, with plain lines; and observation probabilities are shown in three different colours, green, black and violet (depending on the hidden state), with dotted lines.



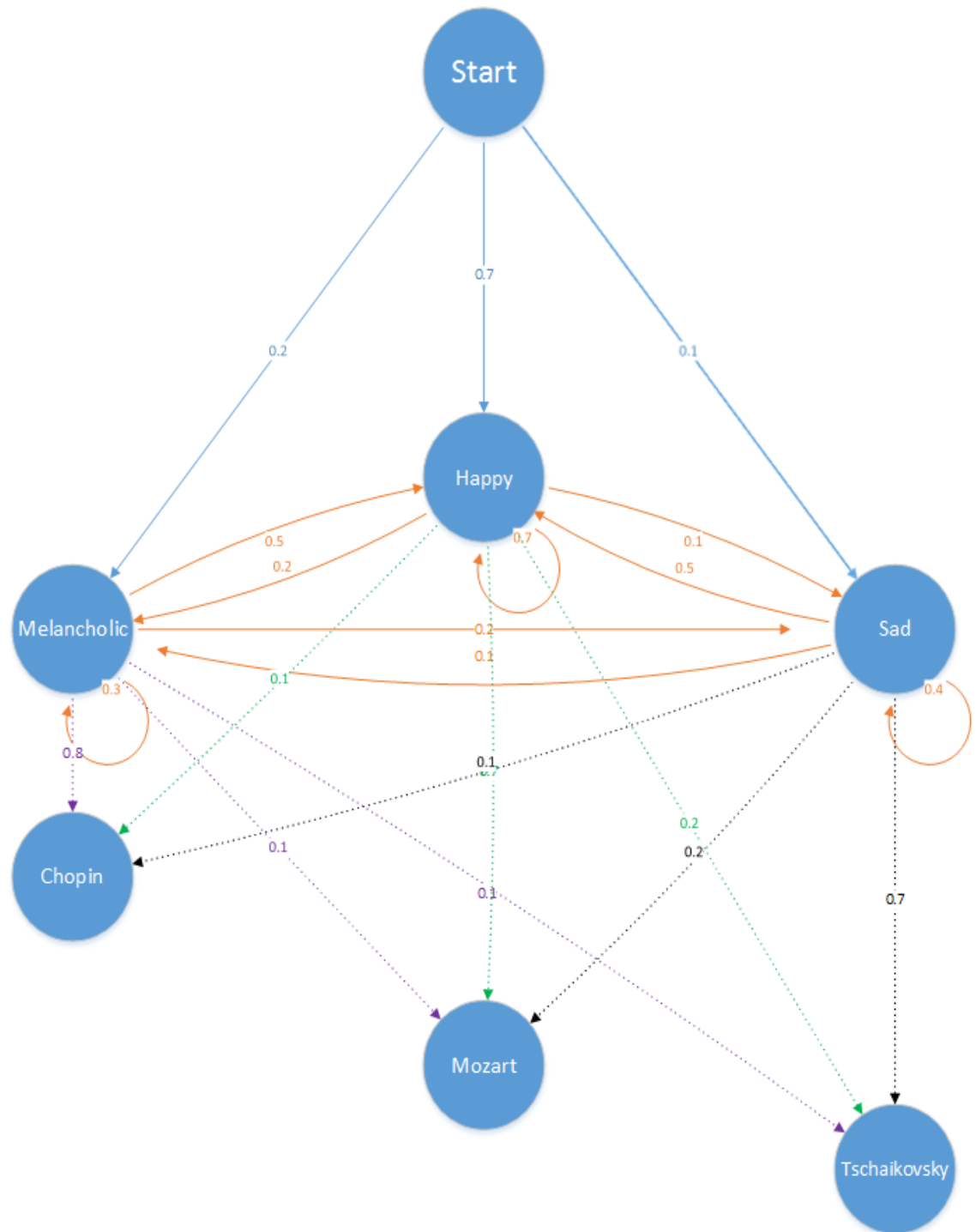


Figure 14: HMM example: a pianist as a neighbour

HMMs can be used in three ways:

**The evaluation problem.** Given the HMM  $M=(A, B, \pi)$  and the observation sequence  $O=o_1 o_2 \dots o_K$ , calculate the probability that model  $M$  has generated sequence  $O$ . In this instance the model is given. The aim is to calculate the probability that the

given sequence  $O$  will be generated. Assuming that we know the pianist's model, what is the probability that the given sequence will be played?

**Decoding problem.** Given the HMM  $M=(A, B, \pi)$  and the observation sequence  $O=o_1 o_2 \dots o_K$ , calculate the most likely sequence of hidden states  $s_i$  that produced this observation sequence  $O$ . In this instance, given an observation sequence  $O$ , the aim is to find the underlying sequence of states that led to the given sequence  $O$ —that is, the sequence of moods experienced by the pianist.

**Learning problem.** Given some training observation sequences  $O=o_1 o_2 \dots o_K$  and general structure of HMM (numbers of hidden and visible states), adjust  $M=(A, B, \pi)$  to maximize the probability.  $O=o_1 \dots o_K$  denotes a sequence of observations  $o_k \in \{v_1, \dots, v_M\}$ . Here, we are given several observation sequences; the question is how to adjust transition probabilities and initial states of the pianist—in other words, how to correct the model of the pianist's choices.

#### 1.2.4. Markov Decision Process (MDP)

Markov decision processes are ‘models for sequential decision making when outcomes are uncertain’ (Puterman, 2009, p. XV). Each model is described as a four-tuple:

**S** – finite state space, with all possible states.

**A** – finite action space.

**P** ( $s''|s, a$ ) – state transition function. This describes how the next state  $s''$  depends on the previous state  $s$  and action  $a$ . The probabilities can be presented in a probability table.

**C** – cost or contribution of taking an action  $a$  in state  $s$ .

At a particular time  $t$ , state  $s$  is a description of the system. From that point it is possible to take an action from a set of possible actions. After that the system may change from state  $s$  to state  $s''$ . The probability of reaching state  $s''$  from state  $s$  by taking action  $a$  is described as  $P$ .  $P$  is a mapping from  $S \times A \times S$  into a real number from 0 to 1. MDPs are connected with the so-called Markov assumption, which states that the next state depends only on the previous state and action, and not on states and actions in the past. Markov decision processes are widely used in risk management, economics, optimization, robotics, and manufacturing.

## 1.3. Music

### 1.3.1. Possible applications

Markov chains have successfully been used in music for the last sixty years. In part, this is because every monophonic melody can be described as a sequence of steps within a finite state space. States can be pitches, chords, rhythmic values, etc. Because of this Markov chains are mainly used in:

- Classification
- Prediction
- Composition

Classification entails using Markov chains to recognize the composer or style of a piece unknown to the user and computer. This application is extensively discussed in (Pollastri & Simoncelli, 2001); (Liu & Selfridge-Field, 2002); (Wołkiewicz, Kulka, & Kešelj, 2007); and (Kaliakatsos-Papakostas, Epitropakis, & Vrahatis, 2001).

### 1.3.2. Prediction

Prediction is usually grounded on an analysis-synthesis theory. To create a new sequence of events one first analyses sample sequences, generalising from the results. Then synthesis is used to create new structures of the same class as the original group of samples. Using this approach can result in problems. For example, the sample may be too small, so that generalisation is not possible. Or the sample members may be so homogeneous that it is not possible to create a new sequence that is not identical with an existing one. The main action is usually *training* of already existing compositions.

One of the first attempts at prediction with Markov chains was made in 1957 by a group of researchers from Harvard University (Brooks et al. 1957). They chose 37 hymn tunes from different composers and centuries. All began on the last beat of a four-beat measure, were four measures long, and contained no durations shorter than an eighth-note. Every duration was notated as summed eighth-notes, so that each hymn could be characterised as a 64-event series. The next step was to conceive each sample as a series of octograms that begin on each successive eighth-note. Thus each hymn yielded 64 octograms, and the entire 37 hymns yielded 2368 octograms. It emerged that only 1701 distinct octograms appeared, with only 1531 distinct heptagrams. This approach allowed them to use Markov chains up to the eighth order.

In synthesising melodies, metrical constraints had to be applied to force the output into the predetermined metrical structure. One rule was, for example, that the two main phrases both had to end on a dotted half-note; another was that the first note of every measure should be struck, not held. These rules acted as gates or sieves: if a note violated a rule, it was discarded and another note was generated. Moreover, if the next fifteen notes did not satisfy the rules the whole hymn was discarded and a new one begun. In the result, after 6000 starts only 600 synthesised hymns were generated.

Example 1 (m=1)



Example 2 (m=2)



Example 3 (m=4)



Example 4 (m=6)



Example 5 (m=8)



Figure 15: Results of the experiment (Brooks, Hopkins, Neumann, & Wright, 1957, p. 181)

Figure 15 presents some results from this experiment. As was predicted, melodies synthesised using a first-order chain ( $m=1$ ) appear to be random; but this decreases as the order increases. For  $m=8$ , the synthesised material was from the same class as the sample. Indeed, further investigation revealed that in most of the synthesised hymns nearly 50% of the melody could be found verbatim in the sample. In a few cases a new hymn was identical to one of the inputs.

This experiment confirmed the problems noted earlier. Using a low order will lead to apparent randomness, but using an order which is too high will produce an output that is extremely similar to the sample. As one of the first experiments to apply Markov chains, this created a basis for new approaches in prediction.

A more recent experiment attempts to overcome these problems (Verbeurgt, Fayer, & Dinolfo, 2004a; Verbeurgt, Dinolfo, & Fayer, 2004b). Instead of using higher orders the aim is to recognize the patterns that occur the most often in the sequence. For example, in a simple melody

**CDEDCDEFG**

The pattern CDE occurs twice, so it is possible to consider it as a characteristic of this melody. On that basis it is possible to create an event space for this melody in which almost every event will be a single note but one event will be the recognized pattern:

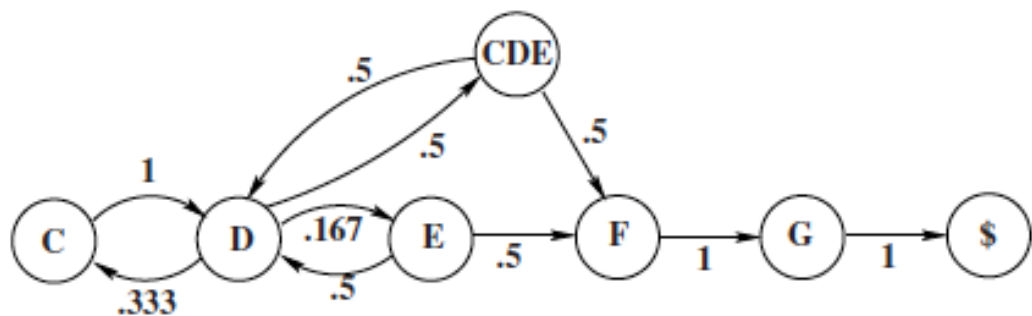


Figure 16: Constructed Markov chain (Verbeurgt, Dinolfo, & Fayer, 2004b)

The results of this experiment are very promising. Figure 17 presents a piece written with Bach's 'Air' as the training material:



Figure 17: A piece written on the basis of Bach's 'Air' using patterns

A contrasting example, produced with a ‘typical’ Markov chain approach, without the use of patterns, is presented in Figure 18.



Figure 18: A piece written on the basis of Bach’s ‘Air’ using a typical Markov chain approach

It is worth noting that the training material (the Bach) was not transposed into a single octave, whereas in the previous experiment it was. This is why in the second example large interval jumps are so common. However, applying the method of pattern recognition helps to avoid this problem, and smooth transitions between notes can be observed. It is also worth pointing out that both pieces respect the common music theory concept of key, in that most of the notes are in the key, with a few accidental notes occurring to create variety.

Both experiments demonstrate the limitations of Markov chains. Because the training corpus consists of existing compositions, low-order chains or limited pattern recognition will produce random results, and high-order chains or extensive pattern recognition will just reproduce already existing phrases.

Different results can be obtained when Hidden Markov Models are used. HMMs are usually used not to predict the entire composition, as in the previous examples, but rather to add another dimension to the existing composition, such as a harmonisation. This too is achieved through training by means of existing music material from a chosen composer or music period.

One of the first attempts was made by Farbood and Schoner (2001), who conducted an interesting experiment in order to create Palestrina-style counterpoint. (However, they do not describe their work as HMM.) A set of species counterpoint

rules in the form of probability tables are implemented, and then the probabilities are estimated from a sample of counterpoint examples. The results are determined by a first-order Markov chain, although sometimes a second-order system is applied. The results of the experiment are very promising; the program was able to generate solutions identical to examples from a counterpoint book (the authors used Jeppesen (1931)).

On the other hand Allan and Williams (2004) proposed a HMM where all the melody notes are treated as visible states, while chords correspond to the hidden states. Then a melody line can be described as sequence of observed states, and a sequence of hidden events creates a possible harmonization for this melody. It is important to note that during training the hidden states of chords and harmonic symbols are actually visible; that is, the learning process is taken directly from observation of the data set. The training process included 121 chorales in major keys and 108 in minor keys; the generated sample covered 81 and 72 chorales in major and minor keys respectively. The two figures show the harmonization of chorale BWV 48 as designed by HMM (Figure 19) and by Bach (Figure 20).



Figure 19: Harmonization using HMM



Figure 20: Original harmonization by Bach

This model appears to be better fitted for prediction purposes: instead of looking at single events, this model takes into account both melody and the surrounding chord. The problems seem to be primarily with voice-leading: large jumps, especially in the bass line, and similar motion in all parts in measures 4 and 7.

Another approach to harmonisation used a Markov decision process (MDP) (Yi i Goldsmith, 2007). Harmonisation in this case is limited to only basic harmonic functions, seven in major scales and thirteen in minor scales. Each state includes two adjacent chords; and because of this, it is possible to evaluate the connection between two successive chords. Harmony is, of course, usually focused on chord sequences longer than two chords, but for practicality a small and manageable MDP was needed. The research may be extended in the future. The choice of the chord is described as an action. If the chosen chord does not conform to the principles of harmony, a penalty is given, which can be described as an increase in cost or decrease in contribution. An abc notation is used in this model, which makes it possible for non-musicians to use the program. In addition, it is possible to generate harmonisations from classical theory or from a specific collection of music, like pieces by one composer or from one music-period. Figure 21 presents an example of harmonisation made with this software.





Figure 21: Harmony generates with MDP

The program is, according to the authors, designed for amateur music lovers. There is a large group of people who would like to harmonise a melody heard on the radio or TV but, because of their lack of theoretical knowledge, are unable to do so. Indeed, the generated harmony is rather superficial, as it is based on a limited range of functions; so the program would be of little value for professional musician. However, an extended version of this approach may provide a very useful tool for more advanced harmonisation.

### 1.3.3. Composition

An interesting use of Markov chains is presented by Charles Ames (Ames, 1989). In this one of the most important features is a waiting probability: the ‘probability that the  $j$ th event of a Markov chain will reside in some state  $k$ , given that the  $j - 1^{\text{st}}$  event also resides in state  $k$ ’ (Ames, 1989, p. 186). Ames used Markov chains in Demonstration 4 (Figure 23), which is one of a series of eleven didactic studies from 1983/1984. In this piece music composed by a computer is presented in a wider context. Demonstration 4 is based on a program with one short loop. The loop generates either a note or a rest. The composing program controls four elements: average duration, articulation, register and chromatic degree. The average duration is calculated on the basis of a four-state matrix, where the states are respectively 2, 3, 5 and 9 expressed in sixteenth notes, while each rest is considered as half of these values. The matrix designed for this element assigns the highest probability to the states on the diagonal; this results in retaining the same note duration for an average time of two measures.

The most interesting detail is that as the note value increases the probability decreases but still remains the highest one.

The second element that the program controls is the specification of rests and notes in the output composition. As in the previous matrix, here also the diagonal probabilities are the highest; in fact they are exactly 0.91 for all possible outcomes. It is worth pointing out that this is the highest probability weighting in all four matrices; because of this it is expected that articulation will be the most consistent aspect of the whole composition. The third factor is register; Ames specified seven registers. To avoid excessive jumps between registers, in this case the diagonal probabilities are again the highest and set at 0.66 exactly. The jumps, if allowed, are skewed to the closest registers. The last parameter that is controlled by the Ames' program determines the chromatic degrees. Those are controlled by a first-order Markov chain, which is presented in the matrix in Figure 22.

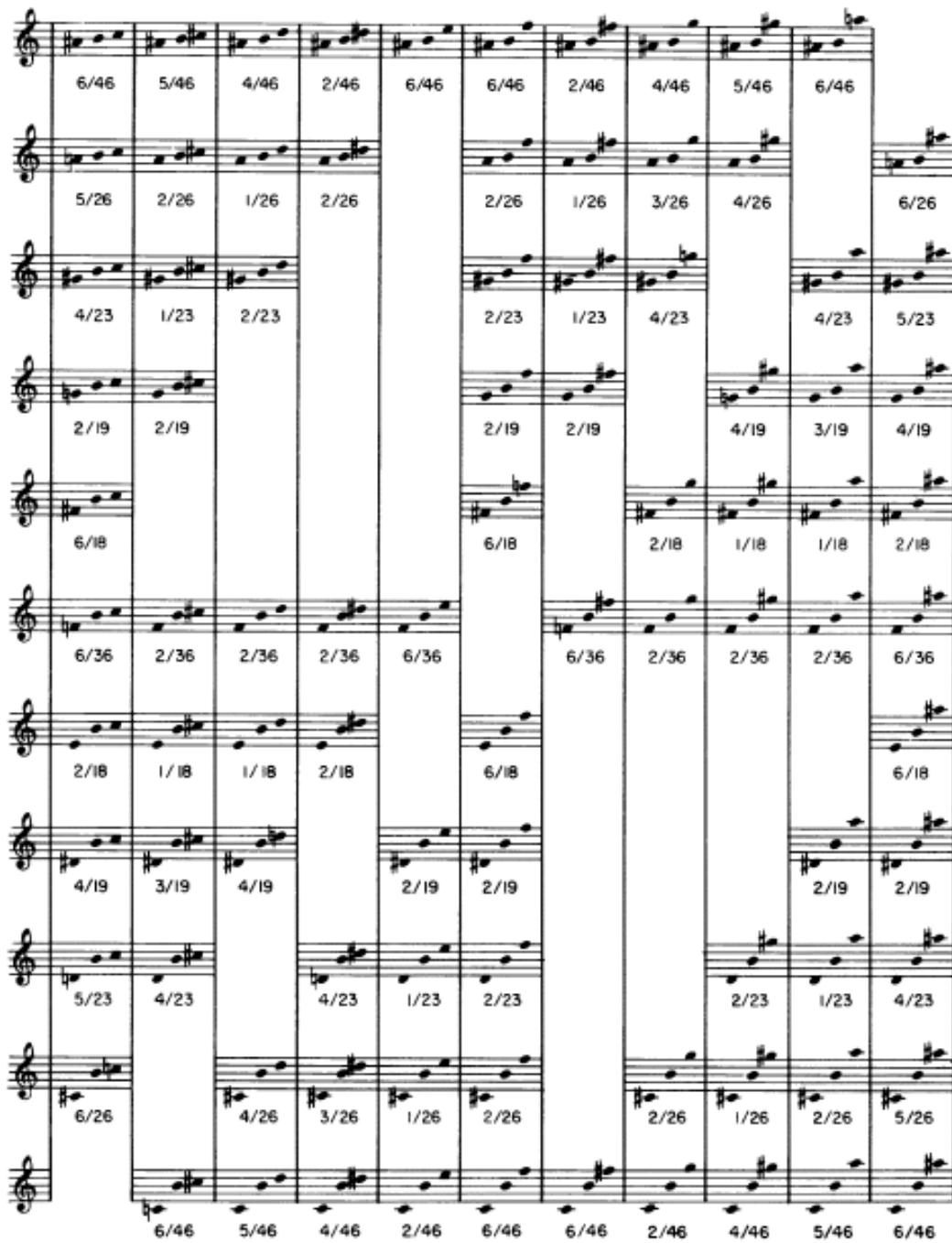


Figure 22: Matrix for chromatic degrees in Demonstration 4 by C. Ames

The outcome of this program is presented in Figure 23. As was expected, articulation changes the least rapidly, while average duration fluctuates in a moderate way.

Clarinet  
STRICTLY  $\text{♩} = 80$

### Demonstration 4

Charles AMES

© Charles Ames 1984

Figure 23: Charles Ames, Demonstration 4

The most interesting results have been obtained when Markov chains are combined with other algorithmic tools. A fascinating approach has been taken by Eduardo Miranda and his research team (McAlpine, Hoggar i Miranda, 1999). They used cellular automata, which are discrete, dynamic systems in that that they change their features over time. Cellular automata are often described as arrays of cells. Each cell can be in one of a finite number of possible states. A specific automata will have an initial configuration that develops over time according to some rules of evolution.

One of the most famous examples is The Game of Life, created by John Horton Conway in 1970 (Gardner, 1970). In this there are four main rules:

1. Any live cell with fewer than two live neighbours dies, as if caused by under-population.
2. Any live cell with two or three live neighbours lives on to the next generation.
3. Any live cell with more than three live neighbours dies, as if by overcrowding.
4. Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

Miranda also used a different cellular automaton, The Demonic Cyclic Space. The evolution rules for this specify:

*A cell which is in state  $j$  at timestep  $t$  will dominate any neighbouring cells which are in state  $j - 1$ , so that they increase their state to  $j$  at timestep  $t + 1$ .*

This automaton is randomized at the beginning, but after a number of steps the cells will self-organize to a pattern. One interesting feature of both The Game of Life and The Demon Cyclic Space is that they both have a toroidal nature, shown in Figure 24. That means that the right edge of the automata space wraps around to meet the left edge and the top edge of the automata space wraps around to meet the bottom edge. For reasons of clarity, however, the automata are presented in planar spaces, where it is possible to track the wrapping of cells in one's own mind.

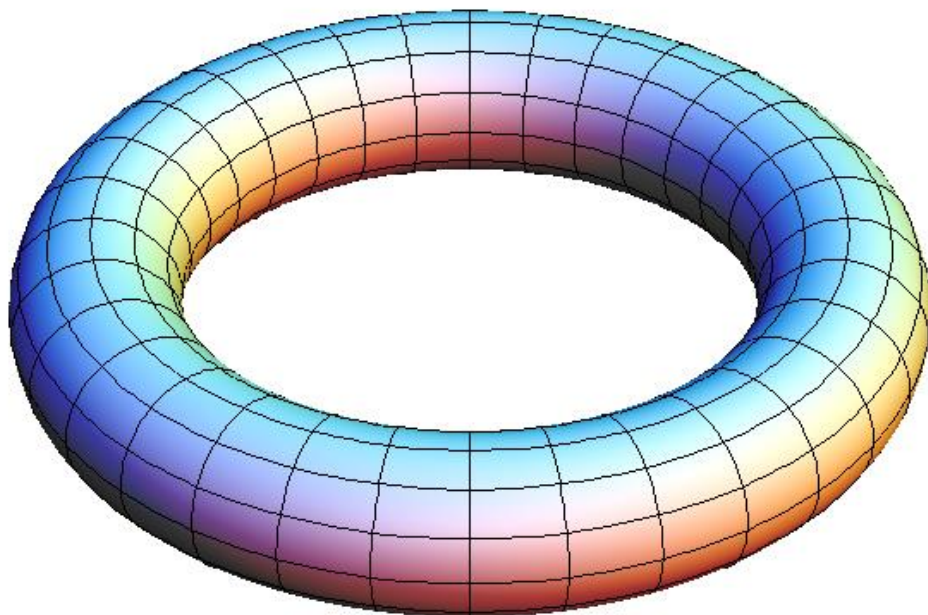


Figure 24: A toroidal space (Virtual Math Museum)

On the basis of these two cellular automata Miranda created CAMUS (Cellular Automata Music). Notes, temporal positions, and durations were created according to neighbouring cells, as in The Game of Life. The main limitation of the original CAMUS was the rhythm generator. The output sequences were usually irregular and difficult to listen to, while note durations sounded completely random. Miranda therefore chose first-order Markov chains to control rhythms. This solution, combined with three-dimensional extensions of cellular automata, is the core of the successor to CAMUS: CAMUS 3D. Three coordinates (x, y, z) from The Game of Life are used to create pitch. As an example, the coordinates in Figure 25 are (5, 5, 2); if these are interpreted as semitones, they create the chord shown. A transition probability matrix, which was used to calculate precise note durations, was created by the composer in the pre-compositional process. This approach represents an interesting conjunction of determinate and indeterminate tools. Automata are wholly determinate: that is, if the initial cells are held constant, the output will be always the same. The reverse is true for stochastic processes like Markov chains: starting from the same transition matrix, varying results will be produced. CAMUS 3D is available on the CD accompanying Miranda's book (2001).

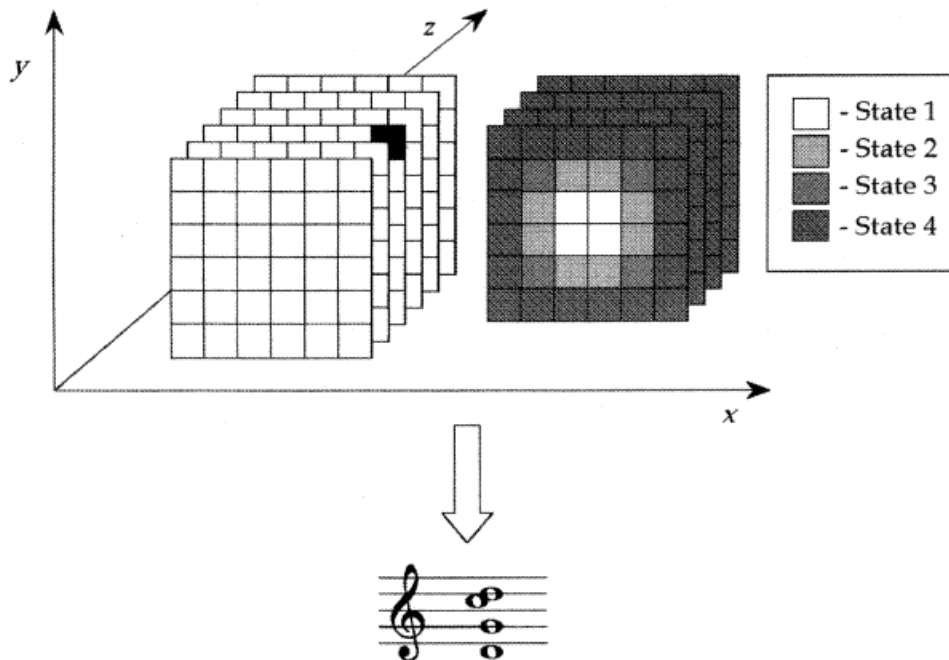


Figure 25: Chord generation (McAlpine, Hoggar, & Miranda, 1999, p. 26)

A different approach was presented by Kevin Jones in his PhD dissertation (1980) and in his later article (1981). In *TEXT YEARS* for choir a special text is prepared to generate distinctive textures. Jones noticed that in the English language certain letters, especially vowels, are read differently depending on their context. For example, one knows how to pronounce ‘o’ only if one knows the whole surrounding context:

Open /'əʊ.pən/

Who /hu:/

Mother /'mʌð.ə/

Moreover, an English native speaker knows how to pronounce even made-up words. In Jones example (1980, p. 100):

*moso; noosoop; kolokos; moalo*

As simple probabilistic solutions were inadequate, according to the composer, a first-order Markov chain was used to create sonic textures. The foundations were short, contrasting fragments of text that were then used to create suitable matrices. To achieve the desired effect a special program was written to facilitate calculation. However, it is important to note that Jones’s program is not applicable only to letters; in fact, one can use any sequence of symbols, including notes.

In another attempt, Jones formulated state space S in an interesting way. In the examples presented in Figure 26, Figure 27 and Figure 28 events vary from single notes

to chords and/or small motives. In this way single states characterise not only pitch but also rhythm and density.

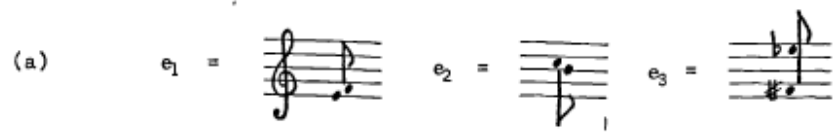


Figure 26: Sample state space (I) (Jones K. , 1980, p. 98)



Figure 27: Sample state space (II), (Jones K. , 1980, p. 112)





Figure 28: Sample state space for clarinet (Jones K. , 1980, p. 121)

Jones does not follow generally accepted terminology; he describes the number of states as the order of the chain and the order of the chain as the dimension of the chain. For example, in Jones's terminology a third-order Markov chain will be described as a 3-dimensional Markov.

## 1.4. Summary

As can be seen, Markov chains have played an important role in twentieth- and twenty-first-century music, and they will continue to do so. The examples presented are only some of the initial attempts to apply Markov-chain concepts to composition and prediction, but they helped establish a basis for a stochastic approach to music. Thereafter larger concepts could be implemented, and these offer new opportunities that combine Markov chains with other processes, determinate and indeterminate, and also with the composer's own creativity. However, the utility of Markov chains, and at the same time their limitation, lies in repeatability. With respect to tonal music, for example, if a high enough order is applied the output will be mistake-free, assuming the provided sample was itself free of errors. But in this case the output will only contain repetitions of already known phrases. One can shape the program to produce pieces without a single mistake, but we will never be able to come up with something new. In this way Markov chains are similar to even the most developed calculation programs.

The definition of Markov chains itself implies one of the possible problems in their application to music.

*Let  $X_0, X_1, \dots$  be a sequence of random variables with possible outcomes  $X_t = x_t \in S$ . Then the sequence is called a Markov chain, if*

*M1. The state space  $S$  is finite or countable;*

*M2. For any  $t \in \mathbb{N}$ ,*

$$P(X_{t+1} = j | X_0 = i_0, X_1 = i_1, \dots, X_t = i_t) = P(X_{t+1} = j | X_t = i_t)$$

In a science like physics or biology, it is relatively easy to use a Markov model for the purpose of describing a well-known process. In fact, the state space  $S$  is usually obvious—for example, the chromosomes  $X$  and  $Y$ , the elements of DNA, etc. The rules underlying the problem being studied are usually well known, and thus it is relatively easy to find appropriate probabilities. Even in the field of linguistics Markov chains seem to give better results than in music. To explain why one must first realise that Markov chains are applied to a specific type of language, one that is defined by a *finite set of rules*. In the next chapter we will consider some of these rules and their implications for music. For the present it suffices to note that music does not always conform to such a clear definition. Thus it is not possible to apply Markov chains to the whole of music. It is not even possible to describe a single state space  $S$ ; and it is impossible even to enumerate the number of spaces needed: duration, pitch, instrument,

or . . . something else? The rules for music composition only apply to a specific period or style; present-day Western music will be regulated differently from music from ancient China. Thus the first steps in applying Markov chains to music must be to define the space(s) and to establish boundaries for the model by setting a finite number of rules within this space. This can only happen if the application is preceded by analysis and prediction within the chosen area, setting constraints according to the musical era or a chosen composer's style. A slightly different approach must be taken if a composer decides to apply Markov chains to create compositions of his own. But this case, too, requires a pre-defined, bounded space with specified rules and elements. Any spontaneous invention of new conditions or the introduction of new elements requires the entire process to be restarted from the beginning.

## 2. Generative grammars

### 2.1. Introduction

In the 1960s linguistics—the scientific study of language—was a special interest of researchers. As a result many new fields were developed. Sociolinguistics was introduced by William Labov and Basil Bernstein (Meyerhoff, 2011), while Systemic Functional Linguistics (SFL) was developed by Michael Halliday (Halliday, 2003). The 1960s are also considered to be the beginning of modern psycholinguistics and the moment when generative grammars were introduced. In 1957 Noam Chomsky wrote:

From now on I will consider a language to be a set (finite or infinite) of sentences, each finite in length and constructed out of a finite set of elements. All natural languages in their spoken or written form are languages in this sense (Chomsky, 2002, p. 13).

This concept was developed over the next few decades and is still an inspiration for other scientists and artists. The novelty of generative grammars lay in the new approach developed by Chomsky. He tried to look at linguistics from a mathematical and logical perspective. Recently Oenbring (2009, p. 93) commented that ‘Chomsky looked outside linguistics proper to formal symbolic logic for a set of methods to ground his concerns in the study of language.’

The formal definition states that generative grammars are grammars designed for the purpose of describing language by means of a set of logical rules. Those rules need to be capable of generating the infinite number of possible sentences of that language and providing them with the correct structural description. Chomsky defined generative grammar in the following way:

By a generative grammar I mean simply a system of rules that in some explicit and well-defined way assigns structural descriptions to sentences. Obviously, every speaker of a language has mastered and internalized a generative grammar that expresses his knowledge of his language. This is not to say that he is aware of the rules of the grammar or even that he can become aware of them, or that his statements about his intuitive knowledge of the language are necessarily accurate. (Chomsky, *Aspects of the Theory of Syntax*, 1969, p. 8)

This linguistic model has developed into a powerful tool widely used in mathematics, psycholinguistics, IT, and music theory and composition. In order to provide a deep understanding of generative grammars and their types some definitions have to be presented.

## 2.2. Basic concept of generative grammars

An *alphabet* is a finite and nonempty set of symbols. For example, the English alphabet contains 26 letters (symbols) and the basic music alphabet contains 7 symbols {a, b, c, d, e, f, g}, while the binary alphabet contains only two symbols {0,1}. The set without any symbols—an empty set,  $\emptyset$ —is not, by definition, an alphabet. However, it is possible to create an alphabet with a single empty symbol, notated as  $\{\epsilon\}$ . A string is a finite and ordered sequence of symbols chosen from an alphabet. The number of symbols within a string is called the length of the string. A finite, infinite, or empty set of strings from an alphabet is called a language.

A formal grammar is a finite set of rules that describe how to generate correct sentences in a formal language. A set of sentences that can be generated by grammar rules constitutes a formal language—if the sentence cannot be created by the grammatical rules it is not a sentence from the described formal language. A grammar can only generate strings; it cannot generate or assign meaning to the generated sentences.

A grammar is formally defined as a tuple  $(V_N, V_T, P, S)$ , where:

- $V_N$  is a finite, non-empty set of symbols called variables or non-terminals, each of which then can be replaced by another non-terminal or terminal symbol.
- $V_T$  is a finite set of terminal symbols, which are unalterable; it is not possible to rewrite a terminal symbol, it terminates any derivation. Each terminal symbol is an actual word from the specified language.
- $P$  is a set of production rules. The production symbol is an arrow  $\rightarrow$ . Each production rule contains a non-terminal symbol that is being described by this rule. This variable is called the head of the production and it is usually on the left side of the arrow. On the right side of the arrow is a string of zero or more terminals and non-terminals, called the body of the production.
- $S$  is the start symbol; it is a specially distinguished symbol in  $V_N$  and is sometimes called the sentence symbol.

Starting with  $S$ , after a finite number of steps according to the production rules, one will achieve a string that contains terminal symbols only. A string is said to be in the language that the grammar generates if it contains only terminal symbols that are derived from the starting symbol  $S$ .

In terms of a formal grammar the alphabet  $V$  is the set of all symbols, terminal and non-terminal:  $V=V_N\cup V_T$ . The set of terminal symbols can be described as the terminal alphabet and set of non-terminal symbols can be described as the non-terminal alphabet. Those sets are disjoint:  $V_N\cap V_T=\emptyset$ . Non-terminals are usually represented by upper-case letters (A, B, C), while terminals are represented by lower-case letters (a, b, c). This nomenclature will be used in the present work.

**Grammar 1**

Consider the following grammar G:

$G1 = (V_N, V_T, P, A)$ , where

$V_n:= \{A, B, C\}$ ,

$V_T= \{d, e, f\}$ ,

**P :**

<b>I</b>	<b><math>A\rightarrow dA</math></b>
<b>II</b>	<b><math>A\rightarrow eCB</math></b>
<b>III</b>	<b><math>B\rightarrow C</math></b>
<b>IV</b>	<b><math>C\rightarrow fC</math></b>
<b>V</b>	<b><math>C\rightarrow def</math></b>

The following generations are then possible (Table 2):

**Table 2: Possible generations of Grammar 1**

<b>Generation</b>	<b>Result</b>	<b>Rule</b>
<b>1.</b>	<b><math>A\rightarrow eCB</math></b>	<b>II rule</b>
<b>2.</b>	<b><math>eCB\rightarrow eCC</math></b>	<b>III rule</b>
<b>3.</b>	<b><math>eCC\rightarrow efCC</math></b>	<b>IV rule</b>
<b>4.</b>	<b><math>efCC\rightarrow effCC</math></b>	<b>IV rule</b>
<b>5.</b>	<b><math>effCC\rightarrow effdefC</math></b>	<b>V rule</b>
<b>6.</b>	<b><math>effdefC\rightarrow effdefdef</math></b>	<b>V rule</b>

As can be observed in the Grammar 1, after the sixth generation the outcome includes only terminal symbols. It can be also seen that the first production rule has been omitted in this example, while the fourth and fifth production rules have been used twice. After six generations a string  $effdefdef$  is achieved. It contains only terminal symbols, all of which belong to  $V_T$ ; hence this string belongs to the language produced by grammar Grammar 1.

If it is possible in grammar  $G$  to derive sentence  $S$  in more than one way then the grammar  $G$  is said to be ambiguous; otherwise it is said to be unambiguous.

Consider the following Grammar 2:

**Grammar 2**

$G_2 = (V_N, V_T, P, A)$ , where

$V_N = \{A, B, C\}$ ,

$V_T = \{d, e, f\}$ ,

$P$  :

<b>I</b>	<b><math>A \rightarrow aBC</math></b>
<b>II</b>	<b><math>B \rightarrow C</math></b>
<b>III</b>	<b><math>B \rightarrow b</math></b>
<b>IV</b>	<b><math>C \rightarrow bC</math></b>
<b>V</b>	<b><math>C \rightarrow c</math></b>

It is possible to achieve the sentence  $abc$  in two ways:

**Table 3: Possible generations by Grammar 2**

<b>I solution</b>		<b>II solution</b>	
<b><math>A \rightarrow aBC</math></b>	I rule	<b><math>A \rightarrow aBC</math></b>	I rule
<b><math>aBC \rightarrow abC</math></b>	III rule	<b><math>aBC \rightarrow aCC</math></b>	II rule
<b><math>abC \rightarrow abc</math></b>	V rule	<b><math>aCC \rightarrow abC</math></b>	IV rule
		<b><math>abC \rightarrow abc</math></b>	V rule

As can be observed, in Grammar 2 it is possible to create the sentence  $abc$  in two different ways. Hence, the described grammar is ambiguous.

Every grammar can be used in two ways: to generate a sentence or to parse an existing sentence. Generation starts from the start symbol  $S$  and chooses one production rule each time there is a choice. Parsing, on the contrary, starts from the bottom level and assigns non-terminal symbols to each terminal according to the production rules. The result of parsing is the starting symbol and a top-level production rule.

The easiest way to present a derivation is by derivation tree, called also a parse tree. Each leaf of the tree corresponds to a terminal, while each internal node corresponds to a non-terminal. While a parse tree shows each derivation, it does not

encode the order they were applied. In Figure 29 the same sentence from grammar 1 effdefdef is derived; however, it is not possible to know what was the order of the generations.

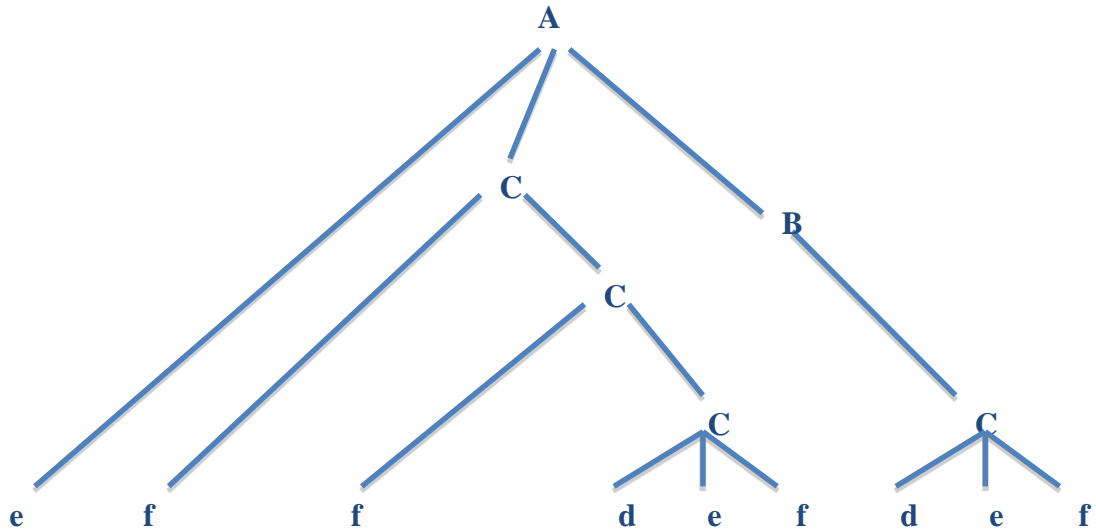


Figure 29: Result of Grammar 1 presented as parse

### 2.2.1. Chomsky Hierarchy

There are four main types of grammars, distinguished by Noam Chomsky (1956). The types present different levels of restrictions and together form *The Chomsky Hierarchy*. Each grammar on each level is part of a grammar on the previous level, as presented in Figure 30. In this way all four types of grammars create one connected set.



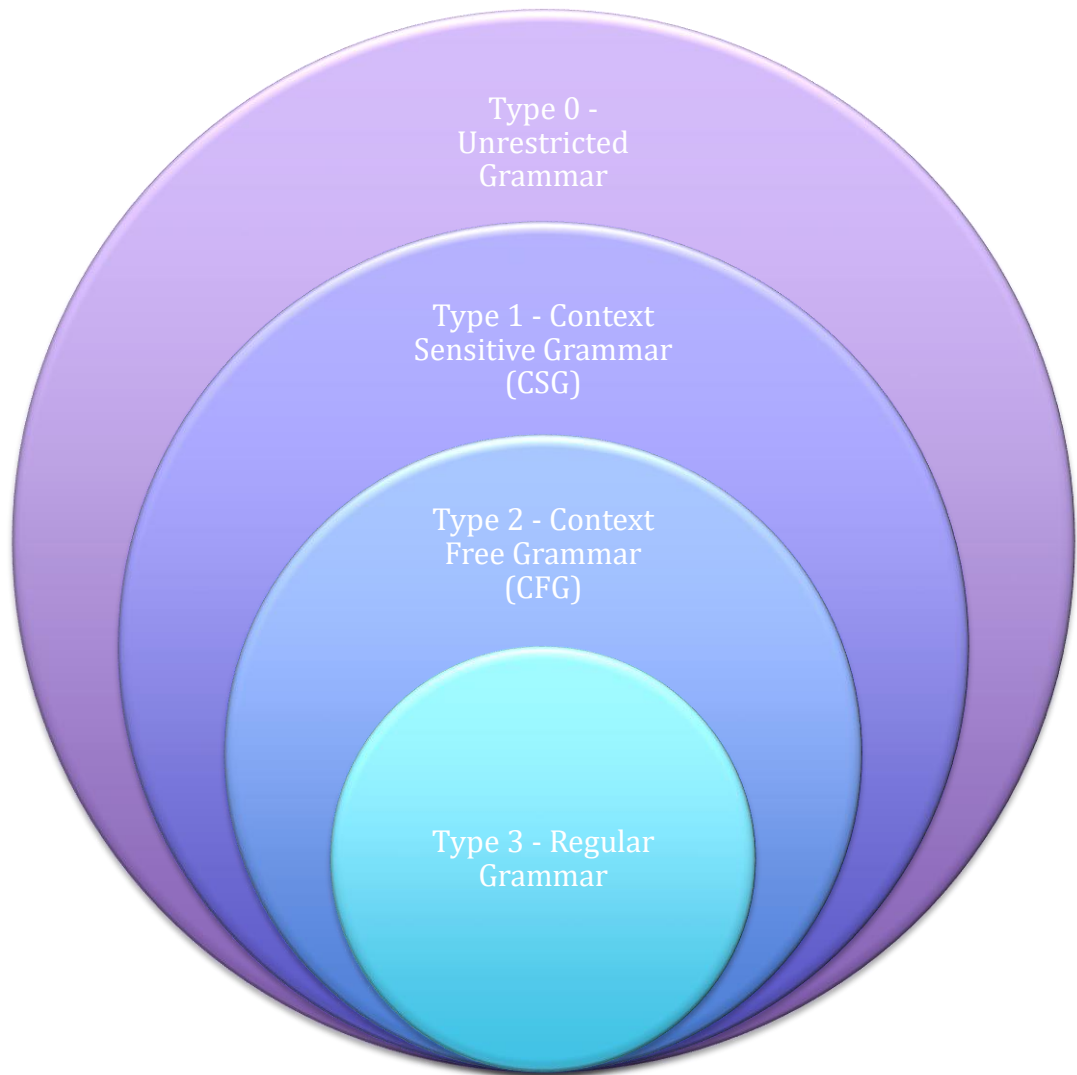


Figure 30: Chomsky Hierarchy

**Type-0 Grammar – unrestricted grammar**

As the name suggests, in this grammar there are no restrictions on production rules. Because of this, each production rule only needs to take the form  $\alpha \rightarrow \beta$ , where  $\alpha, \beta \in (V_T \cup V_N)^*$ . Languages generated by type-0 grammars represent all languages that can be recognised by a Turing machine. Languages created by type-0 grammars are sometimes called recursively enumerable languages.

**Type-1 Grammar – context-sensitive grammar (CSG)**

Each production rule in a type-1 grammar has the form  $\alpha A \beta \rightarrow \alpha \gamma \beta$ , where  $\alpha, \beta, \gamma \in (V_T \cup V_N)^*$ ,  $\gamma \neq \epsilon$  and  $A \in V_N$ . The production rule  $S \rightarrow \epsilon$  exists only if S does not appear on the right hand side of any production rule. A language produced by such a grammar is

called a context-sensitive language. Languages produced by type-1 grammars represent all languages that can be recognized by a linear bounded automaton.

### ***Type-2 Grammar – context-free grammar (CFG)***

Each production rule in a type-2 grammar takes the form  $A \rightarrow \alpha$ , where  $A \in V_N$  and  $\alpha \in (V_T \cup V_N)^*$ . A language produced by such a grammar is called a context-free language. The languages defined by type-2 grammars are recognized by push-down automata.

### ***Type-3 Grammar – regular grammar***

Each production rule in a type-3 grammar can take one of two forms:

- a) Right-linear, where:  $A \rightarrow \gamma B$  or  $A \rightarrow \gamma$ , where  $A, B \in V_N$  and  $\gamma \in V_T^*$ .
- b) Left-linear, where:  $A \rightarrow B\gamma$  or  $A \rightarrow \gamma$ , where  $A, B \in V_N$  and  $\gamma \in V_T^*$ .

While the right-linear and left-linear forms are equivalent to each other, it is forbidden to mix the two within one grammar. The production rule  $S \rightarrow \varepsilon$  exists only if  $S$  does not appear on the right hand side of any production rule. A language produced by such a grammar is called a regular language. Languages produced by type-3 grammar represent all languages that can be recognized by a finite-state automaton.

## **2.3. Grammars in music**

### **2.3.1. Overview**

Based on the concept of generative grammar one can wonder if it is possible to apply similar rules in music. Indeed, according to Steedman, music can be presented as a grammar:

The idea that there is a grammar of music is probably as old as the idea of grammar itself, and the idea that there should be formal grammars of music followed equally hard upon the Chomskian application to natural languages of the formal techniques used to analyse logical and mathematical languages [...] (Steedman, 1996, p. 306).

The possibility of using a grammar to represent music occurred also to Noam Chomsky. In 1979, during the Immanuel Kant Lectures in Philosophy at the Stanford University, he asked the question ‘is music a language?’ and immediately answered it: ‘it all depends on one's definitions, and ultimately it is an unnecessary question; one shouldn't be diverted by it’ (Roads & Wieneke, 1979, p. 48).

The main interest in using generative grammars in music started back in the

1970s. From that period forward this concept has been widely used in algorithmic composition as well as in music analysis.

### 2.3.2. Composing using generative grammars

Consider the following Grammar 3:

Grammar 3

$G_3 = \langle V_N, V_T, P, A \rangle$ , where  $V_N = \{A, B, C, D, E, F, G\}$  and  $V_T = \{a, b, c, d, e, f, g\}$

P consists of the following production rules:

- I.  $A \rightarrow aA$ ,
- II.  $A \rightarrow cdC$ ,
- III.  $A \rightarrow afE$
- IV.  $B \rightarrow bG$
- V.  $B \rightarrow D$
- VI.  $C \rightarrow cD$
- VII.  $C \rightarrow G$
- VIII.  $D \rightarrow dB$
- IX.  $D \rightarrow dF$
- X.  $F \rightarrow acB$
- XI.  $G \rightarrow dABf$

In Grammar 3  $V_T$  contains seven terminal symbols that might be considered as equivalent to notes. Then, instead of composing with notes, a composer is able to compose with relevant production rules within the grammar. The grammar then becomes an easy-to-use compositional tool.

The use of generative grammars as a compositional tool has been developed by many researchers over the past few decades. McComrak (1996) applied string-rewriting grammars based on L-systems into music composition. An extensive survey of the application of generative grammars in music is provided by Roads in his article 'Grammars as Representations for Music' and in *AI Methods in Algorithmic Composition: A Comprehensive Survey* (Fernández i Vico, 2014)

### 2.3.3. Generative Grammar Definition Language

An interesting approach has been undertaken by S.R. Holtzman (1981). He created the

Generative Grammar Definition Language (GGDL) compiler, which is provided with special features designed for musical language. It can be used for both music research and composition. Holtzman based his GGDL on all four types of Chomsky's hierarchy (0-3). In GGDL the simplest production rule takes the following form:

[LHS →A .B .C .D]

This means that the LHS (left hand side) may be replaced by A, B, C, or D. The assignment is random. Holtzman introduces also a *shriek*, which takes the form of an exclamation mark (!) following an arrow:

[LHS →! A .B .C .D]

The above transformation reflects the influence of serialism in requiring that no symbol can be chosen a second time before all other symbols have been chosen.

A third possible rule includes a transition matrix. The generations are then dependent on a row of transition probabilities that states the probability of the production, followed by other possible outcomes that may be generated.

Table 4: Sample LHS transition matrix in GGDL (Holtzman, 1981, p. 52)

[LHS →					
	(A	1	0	1	0)
	(B	0	1	0	0)
	(C	1	1	0	0)
	(D	1	1	1	1)]

When the transition probability equals 1 it means that the given transition is possible, and when the transition probability equals 0 the given transition is not possible. This should not be confused with a stochastic distribution of probabilities. In Table 4 it is possible to achieve C from either A or D; however, the probabilities of those transitions will be the same and, in this case, equal 50%.

Holtzman's compositional procedure includes three stages:

1. Generation: an abstract structure is created based on the rewriting rules.
2. Transformation: the structure is subjected to musical transformations, like inversion or transposition.
3. Mapping: the result of the second stage is then mapped into sounds.

One of the possible applications of GGDL is to produce and describe existing pieces.

In his article (1981, p. 53) Holtzman described a set of compositional rules that was able to produce, among many other structures, the pitch structure of Schoenberg's 'Trio' from *Suite für Klavier*, op. 25 (1925). He also created his own compositions, like *After Artaud*.

#### **2.3.4. Generative theory of tonal music**

Probably the most famous musical application of generative grammars is the Generative Theory of Tonal Music (GTTM). Created by Fred Lerdahl and Ray Jackendoff, this music analysis theory was introduced in 1983. This early approach to the grammatical analysis of tonal music is believed to have had a massive impact on later algorithmic composition. The aim of the authors is to present a theory of music as a 'formal description of the musical intuitions of a listener who is experienced in a musical idiom' (Lerdahl & Jackendoff, 1985, p. 1) By, 'musical intuition' the authors mean the unconscious knowledge that a listener brings to an auditory experience. An experienced listener, who may never have studied music, is able to identify a piece as an example of a specific idiom or genre. Of course such a listener is just an idealisation; it is impossible to find two people with the same ways of hearing music. But the closer a description is to the experience of the ideal listener, the more it can be presumed that the description is of the most 'natural' way to hear a piece.

Researchers through the centuries have tried to trace the connections between music and language, their properties and origins; however, the results, in the form of speculative writings, have proved to be insufficient. The problem was caused in part by attempts to literally transfer linguistic units into music. Researchers tried to find the musical equivalent of nouns, verb, or synonyms. Those efforts were characterised by Lerdahl and Jackendoff as an 'old and largely futile game' (Lerdahl & Jackendoff, 1985, p. 5).

The GTTM is focused on two domains only: pitch and metrical structure. Four types of analysis, all concentrated on hierarchical aspects, are presented:

1. A Grouping Structure organizes the music space into groups. According to the authors this is the most intuitive and basic type of understanding and can be compared to the visual arts. It is also worth pointing out that grouping rules are idiom-independent, which means that even if the listener's knowledge of the idiom is relatively small, it will be still possible to assign grouping structures to pieces in that idiom. The result of the grouping structure analysis is a

hierarchical segmentation of a piece into motives, phrases, periods.

2. The Metrical Structure describes how a listener assigns a metrical structure to a given music space. This type of analysis focuses on stressed and unstressed beats on different hierarchical levels. The result of a comprehensive analysis of this type is a grid, where each row belongs to a distinguished metrical level.
3. Time-span Reduction is mostly based on the information gained from grouping and metrical analysis. Based on this, the importance of individual events is evaluated. The result of this analysis can be presented as a time-span tree, where each event must contain a dedicated head, which is the most structurally important event.
4. Prolongation Reduction is intended to show the tension and relaxation of the musical piece. As for the time-span reduction, here also the result of the analysis is presented in a form of a tree.

All those types of analysis are formalized by three categories of rules (Lerdahl & Jackendoff, 1985, p. 9):

1. Well-formedness rules, which specify possible structural descriptions.
2. Preference rules, which, for any specific experience of a particular piece by a particular experienced listener, define the possible structural descriptions,
3. Transformational rules, which exist to analyse special events, like elisions.

It is worth pointing out that preference rules, which play a major part in GTTM, do not correspond to linguistic theory. On the other hand, transformational rules, which are the core of linguistic theory, are significantly less important. The approach to the rules is one of the biggest differences between the uses of generative grammars in linguistics and in GTTM.

As an illustration, consider the example presented in Figure 31, which contains an analysis of bars 9-16 from the Andante from Haydn's Symphony no 94.

9

Figure 31: Analysis of bars 9-16 from Andante from Haydn's Symphony no 94

The grouping structure is presented on three levels: half-note level, quarter-note level and eighth-note level. However, in bar 16 the half-note level is equivalent to the previous quarter-note level. Indeed, when considering the dynamics and articulation in this particular bar, which can be even named “the title bar”, it is clear that it is not possible to create a larger group than a quarter-note group. In the GTTM, GRP 1 states: “Avoid analyses with very small groups—the smaller, the less preferable” (Lerdahl & Jackendoff, 1985, p. 345). According to this, one pitch in the normal flow of music should not constitute a group. However, GTTM allows exceptions in this matter, when the pitch is strongly isolated or functions as a motive itself. Without doubt both events in bar 16 are strongly isolated, and because neither can be grouped with any other adjacent events they can be treated as an exception. The metrical structure is also presented on three levels, with the whole eight bars as one structure at the highest level.

Figure 32: Time-span reduction

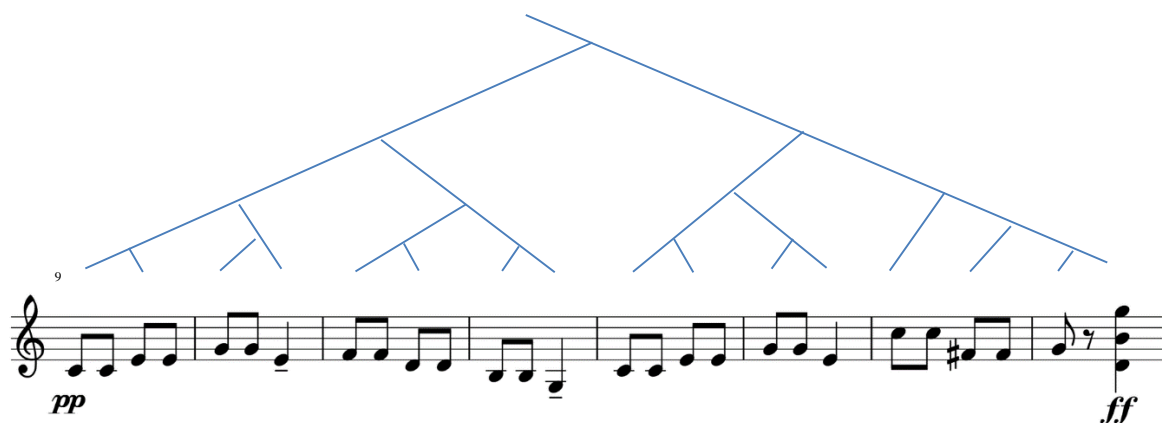


Figure 33: Time-span reduction

The time-span reduction provides the hierarchy of the example, as presented on Figure 32. The event in bar 16 is recognised as being the most structurally important, and thus it is the head of the whole eight-bar phrase.

### Reception

One of the most interesting perspectives on GTTM was provided only recently, in (Katz & Pesetsky, 2011). The authors admit that the theory developed a quarter-century earlier by Jackendoff and Lerdahl still ‘remains the best-developed proposal of its type, unrivaled in comprehensiveness and insight’ (2011, p. 1). They point out, however, that although GTTM was inspired by generative linguistics, its creators admitted that ‘the generative music theory developed here does not look much like generative linguistics’ (Lerdahl & Jackendoff, 1985, p. 307).

Katz and Pesetzky provide an extensive discussion in which they explain that in their opinion music and language are identical in almost every aspect. As has been noted earlier, there are no linguistic equivalents for a dominant chord, nor are there musical equivalents for nouns and verbs; however, the similarity between the two domains resides in their common syntactic components. In other words, music and language differ in their extrinsic construction, but the assumptions and rules behind those constructions are shared between the two domains. This approach allowed Katz and Pesetzky to form the following thesis:

#### *Identity Thesis for Language and Music*

*All formal differences between language and music are a consequence of differences in their fundamental building blocks (arbitrary pairings of sound and meaning in the case of language; pitch-classes and pitch-class*



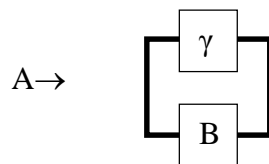
*combinations in the case of music). In all other respects, language and music are identical* (2011, p. 3).

The authors point out the main differences between GTTM and linguistic theory, which they believed were caused by different approaches to formalization and notation that were taken by researchers in those two domains. Afterwards the major similarities are shown—for example, the analogy between prolongational reduction and time-span reduction in music and linguistic syntax and prosody in language—which confirm the hypothesis.

### 2.3.5. Web Grammars

As has been seen, grammars can be used for the analysis of classical music. They can be also very useful tools for composing melodies, but they cannot be used to define vertical relationships in music. This is because of their purely linear structure. To describe both vertical and horizontal relationships **web grammars** are used.

The structure of a web grammar is the same as for a string grammar; it is expressed as a tuple  $(V_N, V_T, P, S)$ . However, the production rules are expressed differently. In a string grammar every production rule has the form  $A \rightarrow \gamma B$ , which express the linear character of the grammar. The rule means that  $\gamma$  and then  $B$  replace  $A$ . In web grammars production rules take the form  $A \rightarrow \gamma/B$ , where “/” means that  $A$  is replaced by  $\gamma$  and at the same time by  $B$ . The vertical character of this type of grammar is easy to see in a graphical representation of the rule, which is called a web diagram:



Instead of the usual vertical result, as was the case for traditional parse trees, this diagram expresses both the horizontal and the vertical aspects of the transition.

Jones (1980, p. 245) noticed that in compositional applications the terminal event space could be expressed as a sequence of traditional notes. With web grammars, unlike string grammars, it is possible to compose music sequences with both vertical and horizontal relationships. Consider the following Grammar 4:

**Grammar 4**

$G_4 = (V_N, V_T, P, A)$ , where

$V_n := \{A, B, C, D, E, F, G\}$ ,

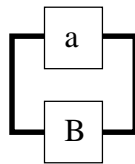
$V_T = \{a, b, c, d, e, f, g\}$ ,

P :

Rule	
<b>I</b>	$A \rightarrow a/B$
<b>II</b>	$A \rightarrow a$
<b>III</b>	$B \rightarrow B/D$
<b>IV</b>	$B \rightarrow bG$
<b>V</b>	$B \rightarrow a/F$
<b>VI</b>	$C \rightarrow Bc$
<b>VII</b>	$C \rightarrow c/f$
<b>VIII</b>	$D \rightarrow d/DE$
<b>IX</b>	$D \rightarrow d/f$
<b>X</b>	$E \rightarrow ACE/e$
<b>XI</b>	$E \rightarrow e$
<b>XII</b>	$F \rightarrow f$
<b>XIII</b>	$G \rightarrow g$

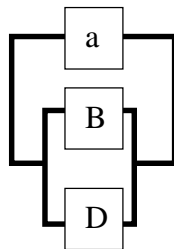
Generation 1:  $A \rightarrow$

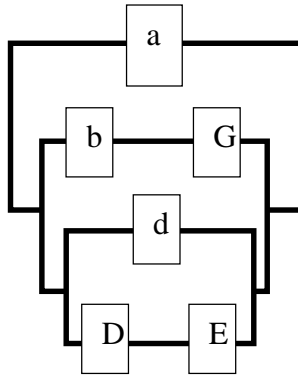
Rules: 1



Generation 2:

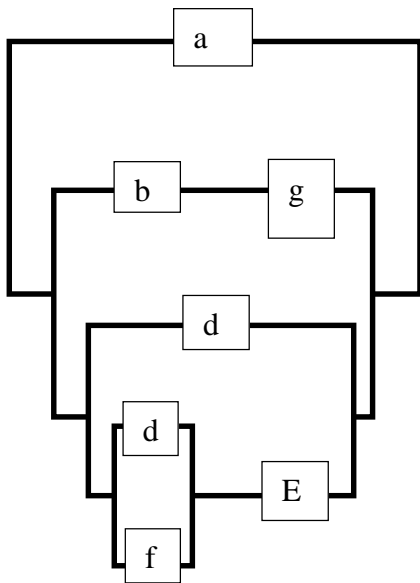
Rule: 3





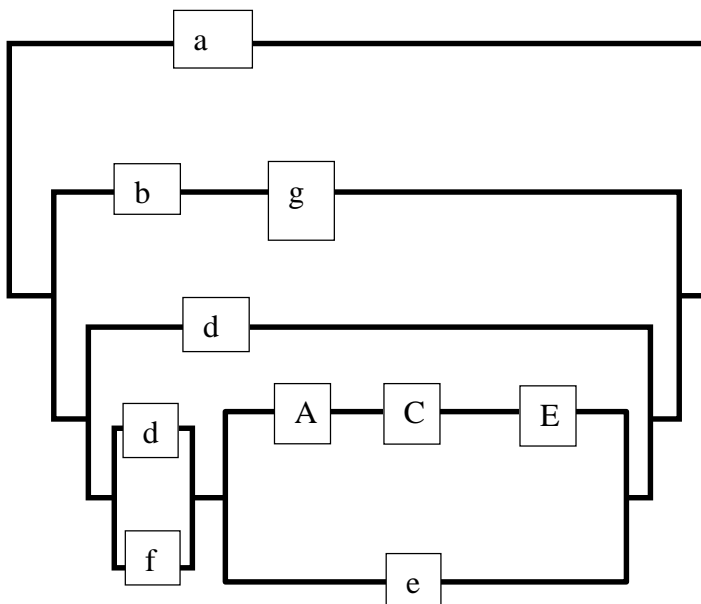
Generation 3:

Rules: IV and VIII



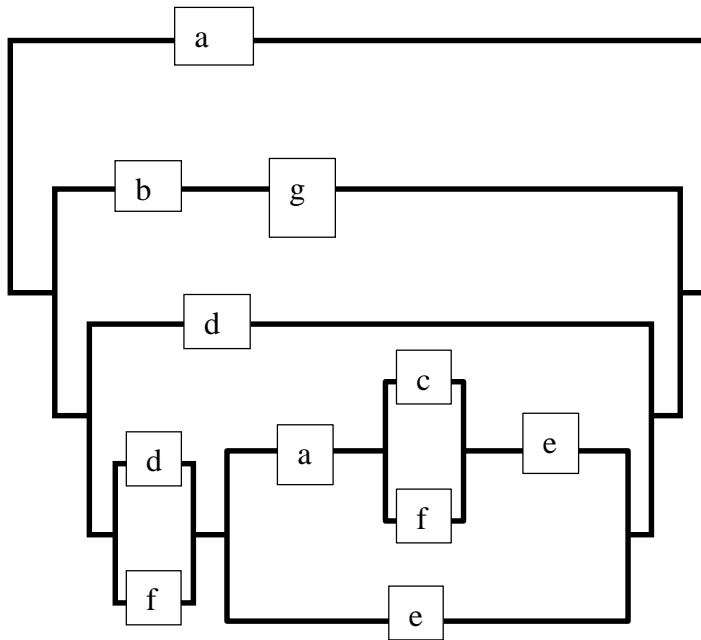
Generation 4

Rules: IX and XIII



Generation 5

Rule: X



Generation 6

Rules: II, VII, XI

A web approach is important primarily because it allows one to create both vertical and horizontal relationships between terminals, which is vital when one uses generative grammars as a compositional tool. The graphical representation of the Grammar 4 in the above example is easily transferable into music, as shown in Figure 34:



Figure 34: Music representation of the outcome of Grammar 4

While earlier, one-dimensional approaches to generative grammars allowed only the creation of melodies, now it is possible to create polyphonic pieces. Another application might be to use another set of terminals—for example, a set of instruments—and create polyphonic music with instrumentation at the same time.

Of course it is possible to create a web grammar with three or more dimensions. A slash with superscripts is then used:  $/^1$ ,  $/^2$ ,  $/^3$ . However, it is generally thought that only grammars of two or three dimensions can be easily created and analyzed.

Web grammars are comprehensively described in (Pfaltz i Rosenfeld, 1969)) and, with relevant music examples, in (Jones K. , 1980) (Jones K. , 1981).

### 2.3.6. Stochastic grammars

Consider the following Grammar 5:

Grammar 5

$G_5 = (V_N, V_T, P, A)$ , where  $V_N = \{A, B, C, D\}$  and  $V_T = \{x, y, z\}$

P consists of the following production rules:

**I.  $A \rightarrow xB$ ,**

**II.  $B \rightarrow yB$ ,**

**III.  $B \rightarrow yC$**

**IV.  $B \rightarrow zC$**

**V.  $C \rightarrow x$**

**VI.  $C \rightarrow D$**

**VII.  $D \rightarrow z$**

Many unique structures may be created on the basis of this grammar. However, one may wonder on what grounds the rules are chosen. It is possible to make an assumption that in all the grammars thus far discussed the probability of applying of each rule was equal:

$$D = \frac{1}{\sum P}$$

However, this equal, random selection from the production rules is sometimes insufficient. If there is a need for a better control of the choice of production rules, stochastic grammars may be applied.

A stochastic grammar is described as a quintuple:

$G = (V_N, V_T, P, S, D)$ ,

where  $V_N, V_T, S$  are defined as in string grammars,  $P$  is the set of production rules, and  $D$  is the set of probabilities associated with production rules. Although stochastic grammars can be created from all four types of grammars in Chomsky's hierarchy, they are mostly used as extensions of context-free grammars (type-2 in Chomsky's classification). They are then called probabilistic context-free grammars (PCFG).

Consider the string given above (Grammar 5).

As presented there, each production rule originating from the same state has equal probability (for example from state B, each rule has probability of 1/3). It is possible to convert this grammar into a stochastic grammar by assigning probabilities to the production rules.

Table 5: Assigning probabilities to Grammar 5

	<b>Rule</b>	<b>Probability</b>
<b>1)</b>	A→xB	1
<b>2)</b>	B→yB	0.2
<b>3)</b>	B→yC	0.3
<b>4)</b>	B→zC	0.5
<b>5)</b>	C→x	0.7
<b>6)</b>	C→D	0.3
<b>7)</b>	D→z	1

An example of a stochastic grammar follows (Grammar 6).

Grammar 6

G6 = (V<sub>N</sub>, V<sub>T</sub>, P, A, D), where V<sub>N</sub>={A, B, C, D}, V<sub>T</sub>={,x, y, z}, D = (1, 0.2, 0.3, 0.5, 0.7, 0.3, 1)

It is worth noting that the usual rule must apply to the probabilities:

$$\sum_{i=1}^7 p_i \in D = 4 = |V_n|$$

Stochastic grammars can be also derived from Markov Chains. Consider the following Markov-chain transition matrix, taken from the ‘Soft Kitty’ example above (chapter 1, figure 3):

	<b>C</b>	<b>D</b>	<b>E</b>	<b>F</b>	<b>G</b>
<b>C</b>		1			
<b>D</b>	1/2	1/3	1/6		
<b>E</b>			2/5	3/5	
<b>F</b>		1/2		1/4	1/4
<b>G</b>			1/2		1/2

The following production rules can be derived from this matrix:

Table 6: Production rules for a Soft Kitty Grammar

<b>I.</b>	<b>C→cD</b>
<b>II.</b>	<b>D→dC</b>
<b>III.</b>	<b>D→dD</b>
<b>IV.</b>	<b>D→dE</b>
<b>V.</b>	<b>E→eE</b>
<b>VI.</b>	<b>E→eF</b>
<b>VII.</b>	<b>F→fD</b>
<b>VIII.</b>	<b>F→fF</b>
<b>IX.</b>	<b>F→fG</b>
<b>X.</b>	<b>G→gE</b>
<b>XI.</b>	<b>G→gG</b>

To allow for termination of this grammar an additional rule must be added:

<b>XII.</b>	<b>D→c</b>
-------------	------------

Using this transition matrix, and taking into account the twelfth rule (of termination), it is possible to create a stochastic grammar having the following form:

Grammar 7

$G7 = (V_N, V_T, P, G, D)$ , where:

$V_N = \{C, D, E, F, G\}$ ,

$V_T = \{c, d, e, f, g\}$ ,

P:

	<b>Rule</b>	<b>Probability</b>
<b>1)</b>	C→cD	1
<b>2)</b>	D→dC	1/3
<b>3)</b>	D→dD	1/3
<b>4)</b>	D→dE	1/6
<b>5)</b>	D→c	1/6
<b>6)</b>	E→eE	2/5
<b>7)</b>	E→eF	3/5
<b>8)</b>	F→fD	1/2
<b>9)</b>	F→fF	1/4
<b>10)</b>	F→fG	1/4
<b>11)</b>	G→gE	1/2
<b>12)</b>	G→gG	1/2

$D = (1, 1/3, 1/3, 1/6, 1/6, 2/5, 3/5, 1/2, 1/4, 1/4, 1/2, 1/2)$

As has been previously noted, the sum of probabilities must equal the number of  $V_N$ :

$$\sum_{i=1}^{12} p_i \in D = 5 = |V_n|$$

### 2.3.7. Problem of termination

Regardless whether grammars are being used for analysis or for composition, it is extremely important to design the appropriate probabilities. A major issue is the termination of the grammar. When the probabilities are assigned in a random or reckless way it is very likely that the grammar will generate structures indefinitely. Consider the following Grammar 8:

Grammar 8

$G_8 = (V_N, V_T, P, A, D)$ , where  $V_N = \{A, B, C\}$ ,  $V_T = \{a, b, c\}$ ,  $D = (1, 0.8, 0.2, 0.6, 0.4)$ .

The production rules consist of the set:

Table 7: Production rules for Grammar 8

	Rule	Probability
1)	$A \rightarrow aB$	1
2)	$B \rightarrow ABAC$	0.8
3)	$B \rightarrow bC$	0.2
4)	$C \rightarrow A$	0.6
5)	$C \rightarrow c$	0.4

Termination is possible after only one generation; the probability of termination then is  $1 * 0.2 * 0.4 = 0.08$ . The next possible termination is after five generations, as presented on Table 8:

Table 8: Problem of termination

Generation	Result	Rules
1.	$A \rightarrow aB$	1
2.	$\rightarrow ABAC$	2
3.	$\rightarrow aBbCaBc$	1, 3, 1, 5
4.	$\rightarrow abCbcbCabCc$	3, 5, 3
5.	$\rightarrow abcbcabcc$	5, 5

The probability of termination in this case is 0.00016384. If termination is not accomplished at this point every further possibility of termination will have a probability approaching 0.

## 2.4. Summary

The purpose of this chapter was to provide an overview of different types of generative grammars and their possible application in music. As can be observed,



generative grammars were a central interest for many researchers in the last century, with Noam Chomsky leading the field. Because generative grammars have been applied mainly in linguistics, they follow Markov chains in their musical uses; nevertheless, it has been shown that they can be successfully applied in music.

While both Markov chains and generative grammars play important roles in the contemporary music, it should be noted that generative grammars have significantly more importance in the field of music theory. GTTM proved to be an important part of twentieth century music analysis, but evolving criticism has led to new developments in recent decades.

For composition, generative grammars have proved to be a useful tool. Their main advantage over Markov chains is that they allow the creation of not only horizontal but also vertical relationships, as was shown in the discussion of Web Grammars. This feature is especially important for composition as it allows for the generation of chords as well as melodies. Combining this with the intelligent design of relevant probabilities can give exceptional results.

It has to be noted that any grammar, even if perfectly designed, will ultimately not replace a composer. A generative grammar can be a great source for diverse and abundant music material; however, decision about the value and quality of what is produced remains the responsibility of the composer.

### **3. Stochastic Composer**

#### **3.1. Introduction**

There are many applications currently available that allow one to use mathematical concepts in the composition process. An interesting website is maintained by Karlheinz Essl; there anyone can access a comprehensive suite of compositional tools (Essl, 2015). Among many available applications, one finds Lexikon-Sonate, an ‘interactive, realtime composition environment for musical composition and live performances’. This software uses algorithms that have been developed by the author since 1985. Essl considers Lexikon-Sonate to be a musical installation that can be also use as a computer instrument for live performances. An interesting feature is that once started, the program can run for years without repeating itself. Another software that is worth noting is Amazing Maze, an application that also uses algorithms created by Essl.

This software can be used to create ‘an astounding sonic cosmos by manipulating instrumental sound particles in time and space’. An interesting feature is the option to run the process either through live user-interaction or automatically.

An important software package is the award-winning Common Music, created by Rick Taube (Taube, 2015). This music composition system ‘transforms high-level algorithmic representations of musical processes and structure into a variety of control protocols for sound synthesis and display. Its main user application is Grace (Graphical Realtime Algorithmic Composition Environment) a drag-and-drop, cross-platform app implemented in JUCE (C++) and S7 Scheme’. It is worth noting that while the first version of the software was created in 1989, the program is constantly improved and updated, with the most recent release in 2014.

FractMus 2000 is another interesting product available currently on the market (Díaz-Jerez, 2015). Created by Spanish pianist and composer Gustavo Diaz-Jerez, it provides a comprehensive set of tools to support composition process. Among twelve algorithms available one finds number theory, chaotic dynamics, fractals and cellular automata. One of the interesting features of this software is the property that each voice can use any algorithm independently from the others.

A good website that provides an extensive list of available applications is Conceptual Algorithmic Music (GDG, 2015). One can find there a list of available programs that support compositional process for Windows, Mac and Linux platforms. While many applications are listed, none of them allow a person to use and combine Markov chains and generative grammars on such a large scale as Stochastic Composer.

Additional software that is already available for use is mostly based on MIDI files, as distinguished from Stochastic Composer, which is based on analysis of musical scores described in more graphical way. Both the generative-grammar and the Markov-chain sections focus on the score itself and on the notes and connections between them. Those note values (Octave, Pitch, Step) are then directly translated into values used in algorithms that traditionally are applied to mathematical variables. A second major difference is that Stochastic Composer abandons the use of MIDI files to represent the music, replacing them with XML files, which are digital representations of musical scores.

One common denominator across the spectrum of such tools is also found in Stochastic Composer: the software does not actually ‘compose’, but serves merely as a

set of tools. These can sometimes be very sophisticated, but they do not usurp the compositional aspects of the work. For example, in FractMus 2000 the composer writes a code in a language defined for the application purpose; similarly, in the Stochastic Composer generative-grammar module, the composer creates production rules that are in fact the programming language of the application. Therefore it is fair to say that in all cases it the computer does not generate music; it generates an answer for a specific set of input variables that are defined by the composer. In most cases the output of the software can then be imported into music-editing software for further processing and analysis and, as all the authors hope, become an inspiration as the composer develops the work.

The scope of this project was to create software that would allow users to compose using Markov-Chain and Generative-Grammar stochastic processes. The main requirement was that the software would have to be easily available to multiple users on various platforms (for instance, on both MAC and Windows computers). Moreover, the software would need to export results to music notation software as well as offering a basic rendering of a composition on the user interface. In order to create a composition the user should need no more than a minimal understanding of the stochastic processes, focusing rather on the potential musical outcome of the software and leaving all mathematical calculations to the digital process itself.

These requirements were satisfied by implementing a solution based on a web application concept that could be delivered to composers through simple web browsers. Using a web browser for software distribution meant that access to the application could also be shared between users, as it could be accessed on users' preferred hardware, even on mobile devices like tablets or smart phones. The software allows a user to further edit the downloaded compositions and use them in any musical notation software that supports .XML file format. Every composition created can be downloaded in the .XML format, which can then be further processed in Finale, Sibelius or other notation packages.

The next section describes the technical aspects of the implementation and deployment as well as the technologies used to achieve the final outcome.

## **3.2. Software design description**

### **3.2.1. Web Application Concept**

The concept of a web application implies creating client-server applications in which the client side of the application runs in the web-browser. This means that the user interface for the application is delivered to a user via an internet connection, but the core algorithms are executed on a server.

Using such an approach gives a developer the convenience of using a web-browser as a medium and saves the need of distributing and installing software for the end users. In the case of Stochastic Composer, that approach allowed easy distribution to both MAC and Windows users without the necessity of writing or duplicating any code. The Stochastic Composer follows a classical structure of dividing the web application into the following tiers (Figure 35):

- Client Layer – user interface
- Presentation Layer – server access layer, by which the user interface sends commands to the server
- Business Logic Layer – implementation of all the algorithms and data processing
- Data Access Layer – implementation of database access
- Database – storage for users and their compositions (data)

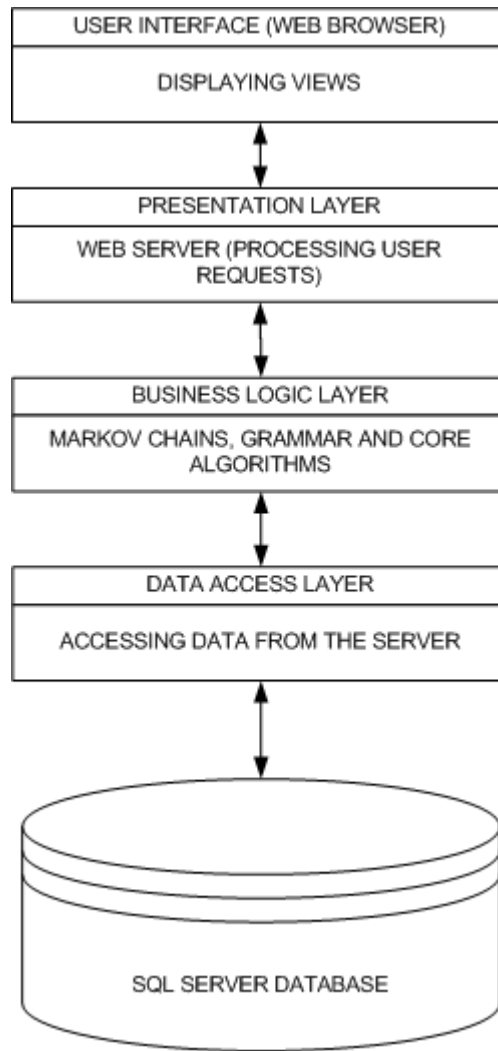


Figure 35: Stochastic Composer - Software Structure

Following that structure allowed changes and development to be carried out without disturbing the end-user. Every time the end-user accesses the application the browser obtains the new version from the server. That means that any changes to the Client Layer can be carried out and implemented without affecting the functionality of the server and the database; changes to the algorithms or the database do not affect clients and, thanks to delivery via a web-browser, no update is required.

### 3.2.2. Server side

As mentioned in the previous paragraph the key elements of the software reside on the server. Stochastic Composer based its server side on the .NET Framework, a solution developed by Microsoft that is a key component of all modern windows-based software solutions. Version 4.5.2 was used for the main web applications and

accompanying software libraries, which were coded using C# as development language.

The whole server side, consistent with one of the .NET Framework core principles, is based on an object-oriented programming (OOP) model, which organises the code around objects and classes. Objects are very often real-life data examples that contain both data and procedures that can be executed on those objects. Classes are definitions of the data types and methods that are used to create further objects.

In case of Stochastic Composer the simplest example is a Musical Score (class -> score) that has a Title (a simple property) and multiple Parts (list of part objects). A Part is, in turn, built of measures (objects), each of which is created of multiple measure elements (notes, rests, etc.).

The server side was built from a few components that form the core of the Stochastic Composer application:

- Domain – The domain class library contains class definitions for all relevant objects used in the software. Classes define the music notation (notes, pitch, score, measure, etc.) and also include software-oriented classes (Markov-chain nodes and transitions, or grammar production rules). In the domain library, the structure of all those classes is defined, but no actual code is executed.
- Data Access – The data access library contains database access definitions and database formulas for creating the database using a ‘Code First’ approach. Such an approach creates the whole database structure during the first run of the software and allows for easier maintenance of the database in line with the OOP concept. In addition, the data access library contains all the static helper classes (for example, the ‘MathHelper’ class contains a definition of the Gaussian Kernel used to assign notes to a specific octave).
- Grammar Manager – This library is the core of the business logic for generative grammars. Its classes contain procedures for creating and maintaining production rules and for their validation, as well as for the generation of all types of grammars supported.
- Stochastic Composer – This is the server application that integrates all class libraries and provides the business logic and access for all application components. This is the brain of the whole software package, as it is responsible for hosting all the server tiers of the application. It is built based on ASP.NET,

which is responsible for serving data to the client sessions as well as executing code from the Grammar Manager and Data Access libraries. This section also hosts all client-based code for the User Interface, which is served to the user's web-browser through HTTP protocol.

- Unit Test – This is the last library included in the project and contains the code that is used purely for testing the server-side procedures.

To summarise, all the components of the server application combine to create a core structure on which the whole software is based.

### **3.2.3. Database**

Microsoft SQL Server is used as a database for Stochastic Composer. That database (Figure 36) is used to store:

- User data (passwords [in coded format], usernames, and privileges);
- Composition data (XML outcomes, titles and descriptions) for both created and uploaded compositions;
- Grammar or Markov-chain details that allow for analysis of compositions created using the software.

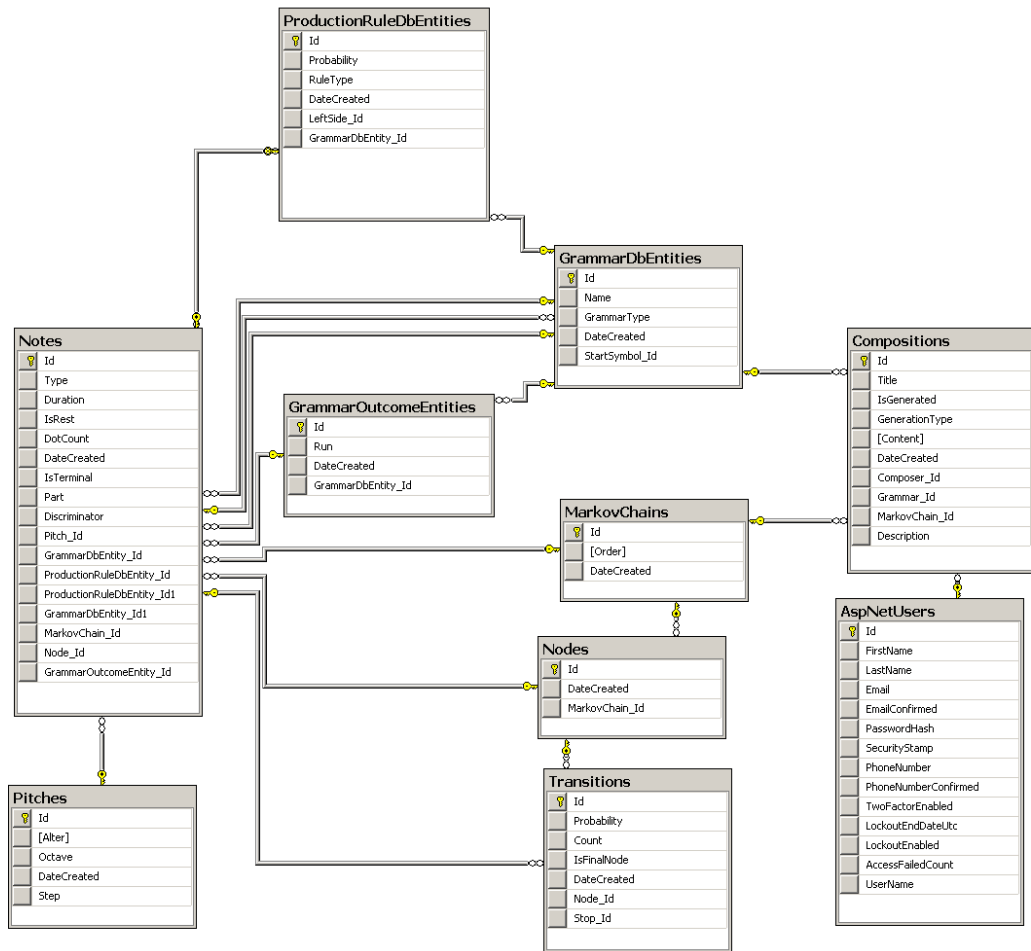


Figure 36: Stochastic Composer - Database Structure

Storing compositions on the server is important and allows the author of this thesis to analyse outcomes generated by other composers as well as permitting the composers themselves to see and download the pieces composed by use of the software. The database stores all user information as well and allows for maintenance of user permissions and access to compositions created by them, simultaneously protecting access to other composers' work.

### 3.2.4. User Interface

As previously described, the user interface was delivered to the user by means of a web browser. The development of the web-browser interface required implementing two core functionalities:

- Serving views to the user where requested
- Executing user actions within the browser before sending data back to the server

In order to develop a fully functional user interface and deliver it via a web browser the traditional HTML had to be extended by providing dynamic content and a capability



for handling complex data structures (compositions, chains, grammars). The user interface was therefore based on AngularJS, a software framework developed originally by Google that allows for creation of single-page applications (web pages that allow a user to navigate through them without the need to refresh the browser view).

Using the above tools allowed the author to create a fully working user interface that operates within the end-user's browser. By using SPAs (single-page applications), which provide a user experience more like a desktop, a user feels more like operating a normal software application, rather than a navigating a web page.

In addition, the user interface is designed to be responsive, which means that interface elements change position, shape, and size to fit different size screens, making it ideal for mobile devices like smart phones or tablets.

### **3.2.5. Deployment**

The Stochastic Composer was deployed in Microsoft Cloud (previously named Microsoft Azure) as a web service. Deployment in the cloud allows multiple users to be given instant access to the software. The application was deployed under the domain <http://stochasticcomposition.azurewebsites.net>.

Deploying the application in the cloud also guarantees uninterrupted access, compared to deployment on a home-based server, and immediate exposure on the internet.

### **3.2.6. MusicXML**

One of the biggest challenges in developing the software was choosing the format in which the compositions generated by the software can be downloaded and, more importantly, displayed on the web page.

The key problem was that the generated outcome had to be available to multiple users with different music backgrounds, using multiple and varied music-notation applications. After in-depth research MusicXML was chosen as the main distribution method. MusicXML was designed for sharing sheet music files between applications. This relatively new standard is supported by over two hundred different music applications and makes it possible to easily share sheet music among composers and artists. The software accepts and generates music scores in accordance with version 3.0 of the format.

The simple melody presented in Figure 37 consists of one measure and four notes. The XML file (Figure 38) contains a heading that summarises the document type. The `<score-partwise>` modifier starts a new score. Every XML field has to have a closure, so at the end of the file the score also ends with `</score-partwise>`. This XML standard is consistent across all files.

This particular file has one part that contains one measure with 4 notes (G, F, E and D). Every component of the score that is required to reopen this score properly in different software is therefore saved into an XML file, which is stored in the database and can be downloaded by the user.

The basic components are presented below:

- `<note>` – Note structure
- `<pitch>` – part of the note describing pitch (contains `<step>`, `<alter>` and `<octave>`)
- `<type>` – type of the note , for example ‘whole’, ‘16th’



Figure 37: MusicXML Example - Score

```

<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE score-partwise PUBLIC "-//Recordare//DTD MusicXML 3.0 Partwise//EN"
"http://www.musicxml.org/dtds/partwise.dtd">
<score-partwise version="3.0">
  <part-list>
    <score-part id="P1">
      <part-name>Melody</part-name>
    </score-part>
  </part-list>
  <part id="P1">
    <measure number="1">
      <attributes>
        <divisions>256</divisions>
        <key>
          <fifths>0</fifths>
          <mode>major</mode>
        </key>
        <time>
          <beats>4</beats>
          <beat-type>4</beat-type>
        </time>
        <clef>
          <sign>G</sign>
          <line>2</line>
        </clef>
      </attributes>
      <note>
        <pitch>
          <step>G</step>
          <alter>0</alter>
          <octave>4</octave>
        </pitch>
        <duration>64</duration>
        <type>quarter</type>
      </note>
      <note>
        <pitch>
          <step>F</step>
          <alter>0</alter>
          <octave>4</octave>
        </pitch>
        <duration>64</duration>
        <type>quarter</type>
      </note>
      <note>
        <pitch>
          <step>E</step>
          <alter>0</alter>
          <octave>4</octave>
        </pitch>
        <duration>64</duration>
        <type>quarter</type>
      </note>
      <note>
        <pitch>
          <step>D</step>
          <alter>0</alter>
          <octave>4</octave>
        </pitch>
        <duration>64</duration>
        <type>quarter</type>
      </note>
    </measure>
  </part>
</score-partwise>

```

Figure 38: Music XML - Simple Example

MusicXML is therefore a universal sheet music distribution method, and it was successfully integrated into Stochastic Composer.

### 3.3. Algorithm Implementation

#### 3.3.1. Music Notation

One of the more important components of the Stochastic Composer was presenting music notation in a form that could be used for stochastic algorithms. Notes are not directly translatable into values that traditionally can be used in Markov Chains or in Grammars, and because of this some artificial restrictions were created.

The diagram in Figure 39 presents a basic class diagram of all the key music-notation components. At the centre of the algorithms the class Note was used as an entry-level object that is adjusted and processed for use across the whole project. It is important to remember that Note is not reserved only for sounded pitches but can also represent a rest (by setting the IsRest field to true).



Figure 39: Stochastic Composer - Music Notation Classes

- Markov Chains and Generative Grammars each required a different approach in analysing what Note parameters are used in algorithmic calculations:
  - a Markov Chain uses most of the note parameters to create a Node; those include:

- Type of Note (including rest)
- Step
- Alteration
- Octave
- Number of Dots
- Generative Grammars required a much more conservative approach and used only:
  - Step
  - Alteration

### **Markov Chains**

The Markov-chain implementation contains three major classes:

- MarkovChain – the main container for all object components (Nodes and Transitions)
- Node – the class holding Node information, including notes forming the Node and all Transitions, with their probabilities
- Transition – an object holding information about the Start and Finish Node, as well as the probability of the transition

There are two main methods applied in the Markov-chain algorithm:

- Chain Generation
- Prediction/Simulation

Figure 40 presents a simplified flowchart for a chain generated from a melody. The generation algorithm creates nodes and transitions between the necessary nodes and assigns probabilities to the transitions it creates. Transitions have to be recalculated after every note as the total value for the connection changes. The number of notes used in each node is determined by the Order value, which is provided by the user. After a whole melody is processed an additional transition to a ‘Final’ note is created, and the chain is generated.

In Figure 41 one can see the prediction/simulation algorithm. The only difference between predicting and simulating is in the selection of the first node. In prediction, the whole melody is added to the outcome, which means that the last node will complete the current melody. The generation algorithm uses a random number generator, which

chooses a value from 0 to 1. That value is then mapped to a range, and the transition which is assigned to that range is selected. Such an approach is possible because the sum of probabilities always equals 1, and the size of each range is proportional to the probability value.

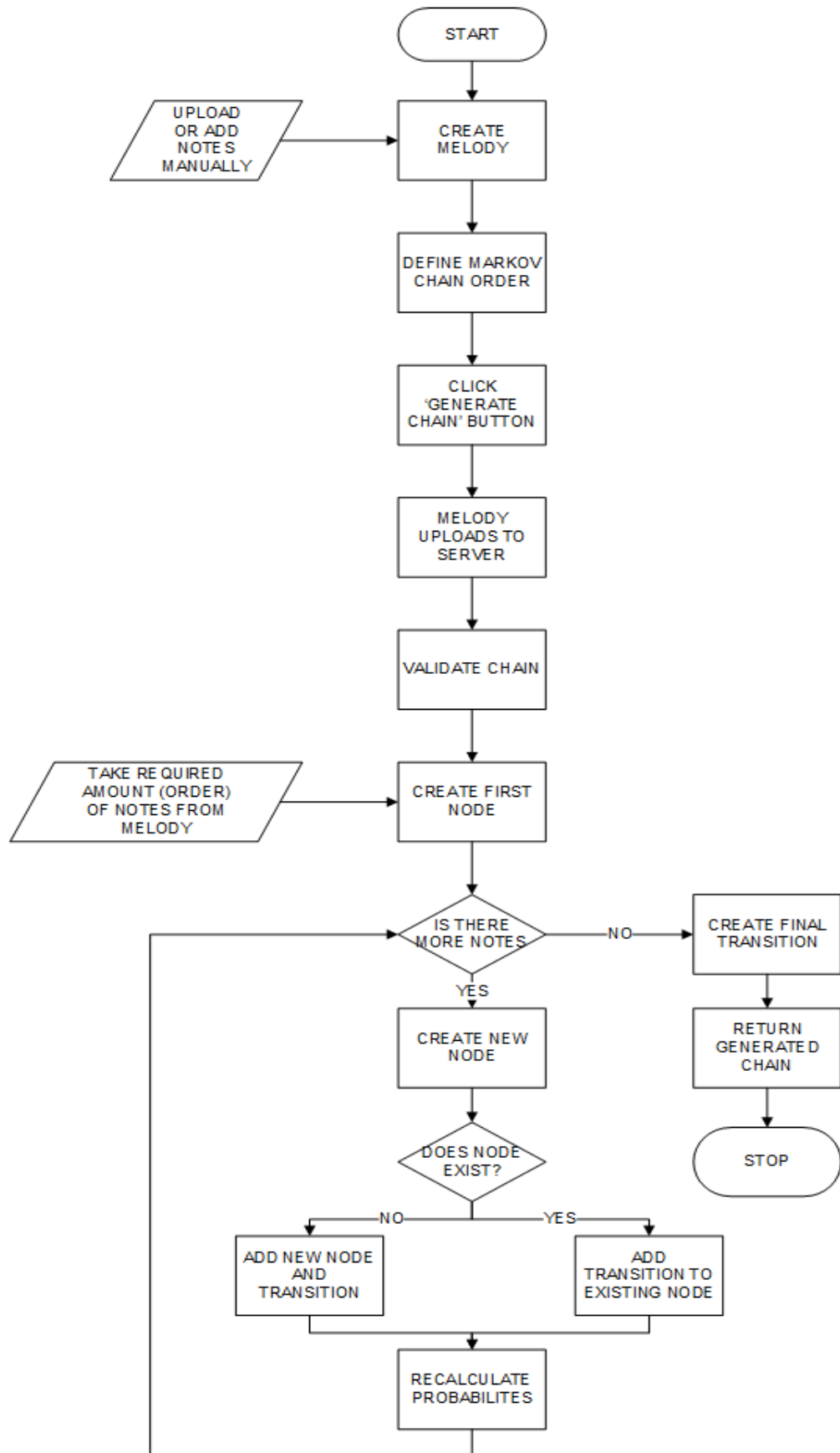


Figure 40 - Stochastic Composer - Markov Chain Generation Algorithm

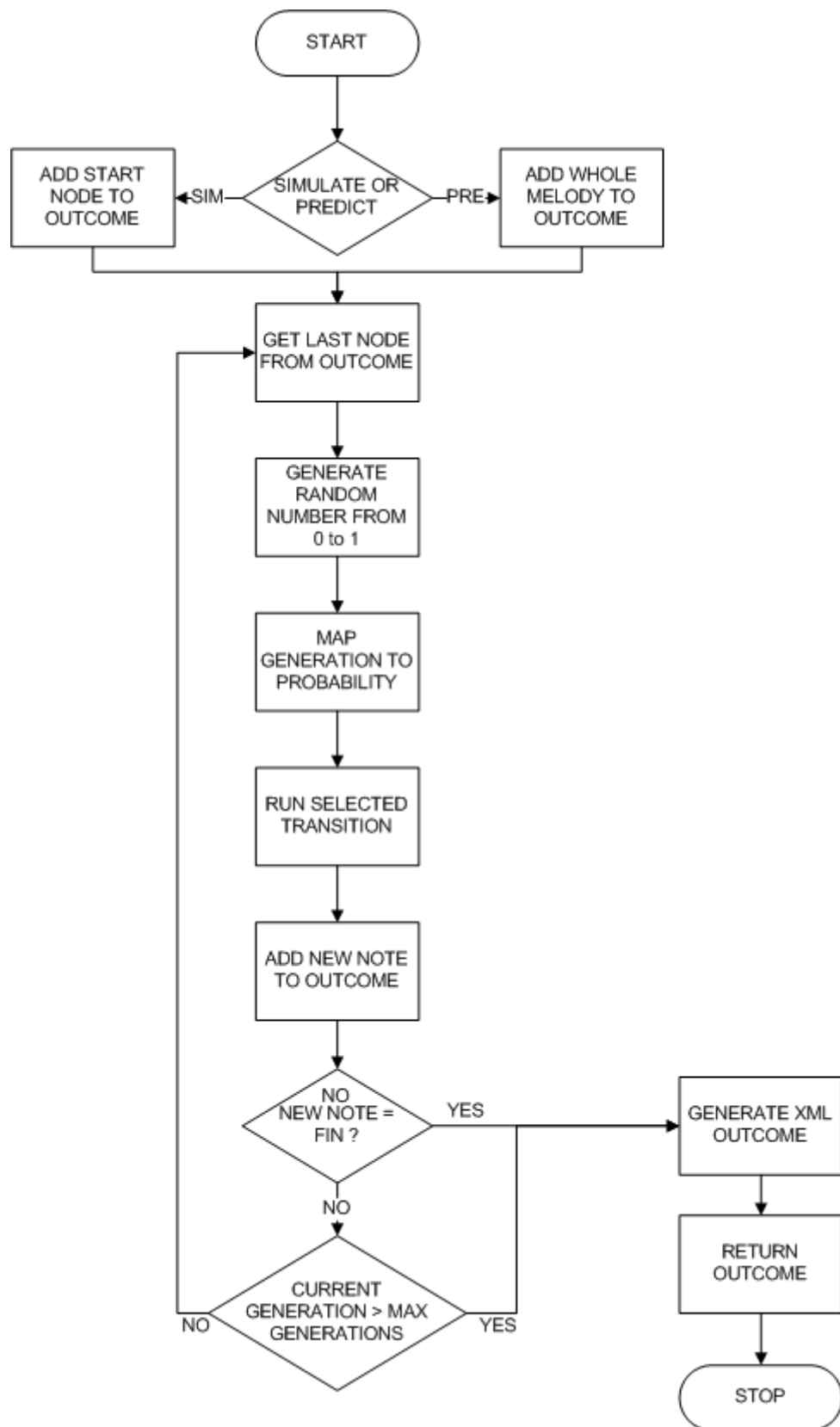


Figure 41 - Stochastic Composer - Markov Chain Prediction/Simulation Algorithm



### 3.3.2. Generative Grammars

Generative grammars are implemented in a way that allows for three types of grammars:

- Linear
- Stochastic
- Web

Each type follows the same implementation structure:

- Grammar – the main container for all object components
- ProductionRule – a class holding notes forming Left, Right and Conditional Right sides (for Web Grammar only)

There are two major operations that are carried out in generative grammars:

- Validation
- Outcome generation

Validation of the grammar is executed before the user is allowed to run the grammar; it checks that the following conditions are met. Grammars that are not valid cannot be used, since the outcome wouldn't be populated.

- Check that a Start Symbol is defined
- Check that all Production Rules have the Right Side defined
- Check that all non-terminal notes have at least one production rule defined
- Check that there is at least one terminal note defined
- Check that in the right side of the production rule non-terminal notes do not appear after terminal notes
- For Stochastic/Web Grammars an additional check of the sum of probabilities is made. (The sum of all probabilities for a given non-terminal note must equal 1.)

The outcome generation algorithm, presented in Figure 42, uses an approach similar to the Markov-chain algorithm, in that it uses a random number generator for selecting a rule to be applied in the next generation. Replacing all the symbols in a single generation triggers a series of checks to confirm that a further generation is required. In addition, for Web Grammars, the algorithm is more complex, since a proper search through grammar generations is required; so a Binary Search Tree is created, which guarantees that every symbol in the grammar will be accessed in a given generation.

The Max Generation parameter is used to prevent the algorithm from entering into an infinite loop.

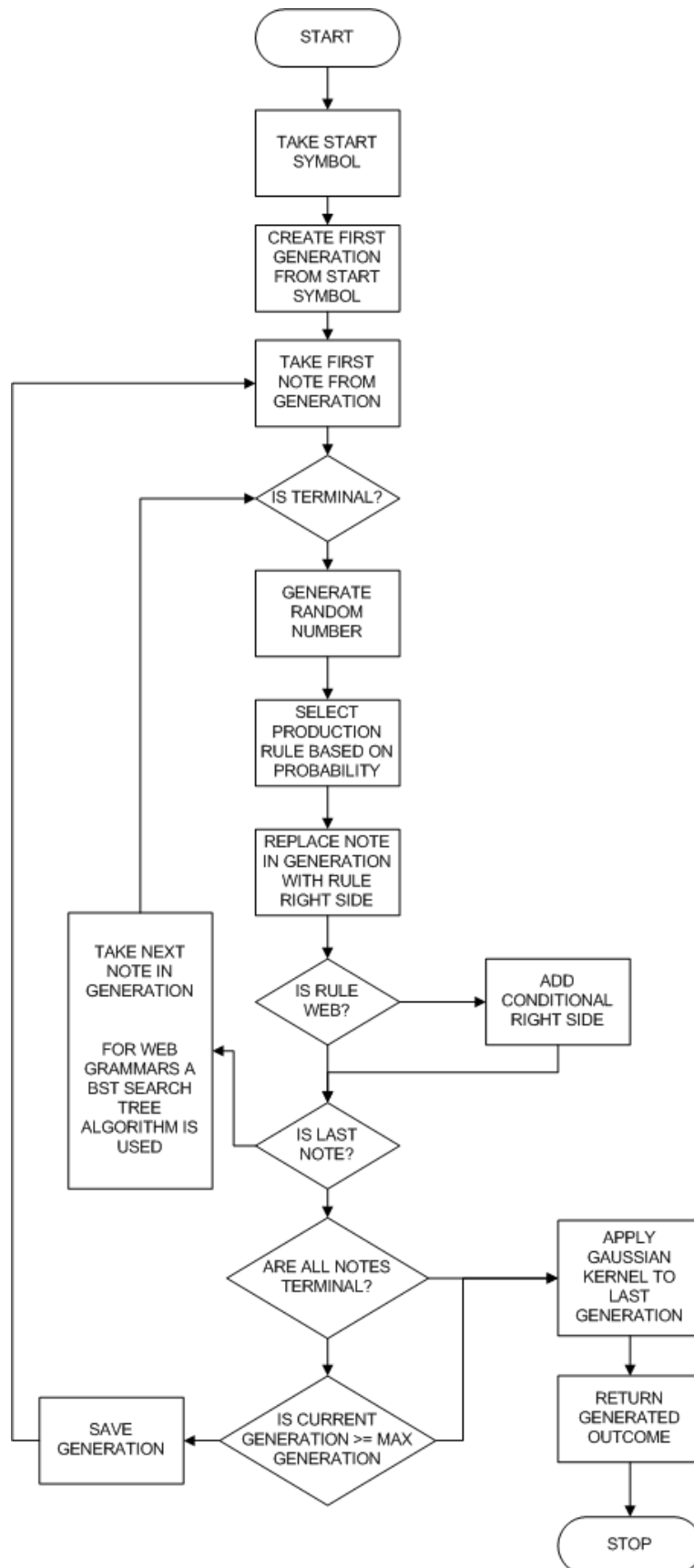


Figure 42 - Stochastic Composer - Grammar Outcome Generation Algorithm

At the end of the algorithm, the last generation is modified by a Gaussian Kernel selection to determine the Octave for each of the Notes. The user selects the focus of the Gaussian Kernel, which is a discrete version of the Gaussian distribution.

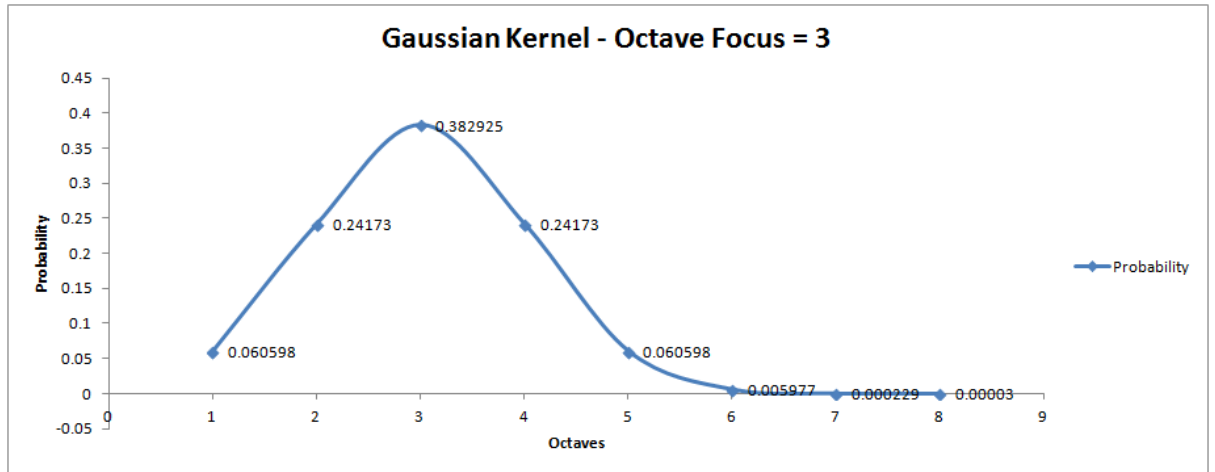


Figure 43 - Stochastic Composer - Gaussian Kernel Example

Linear and Stochastic grammars generate a Music XML outcome, but web grammars are significantly more complex to be presented as part of a score, so only a text output is returned.

### 3.3.3. Navigation

To use the program one needs to register on the website by providing a username, password, and email address. It is also necessary to agree that generated compositions might be used for the data in this thesis.

After registration and login in a welcome screen appears (Figure 44).

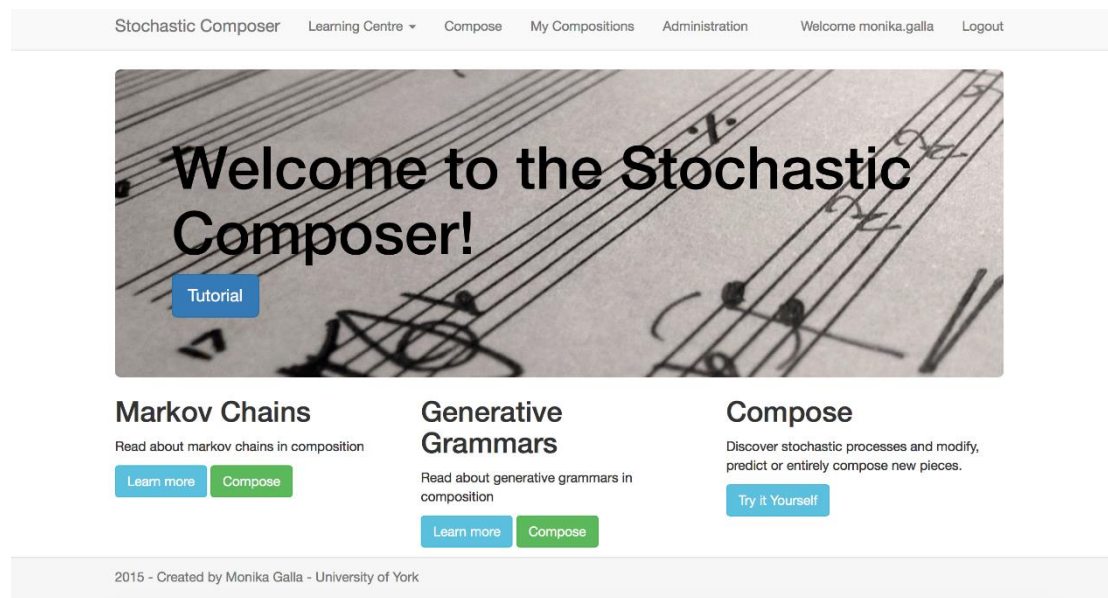


Figure 44: Welcome screen in Stochastic Composer

There are three main tabs offered to the user:

- Learning Centre
- Compose
- My compositions

It is also possible to navigate from a bottom panel, where can be found buttons that redirect the user to specific Learning Centre and Compose sections.

By clicking on the first tab – Learning Centre – the following options can be found (Figure 45):

- Markov Chains
- Grammars
- Tutorial

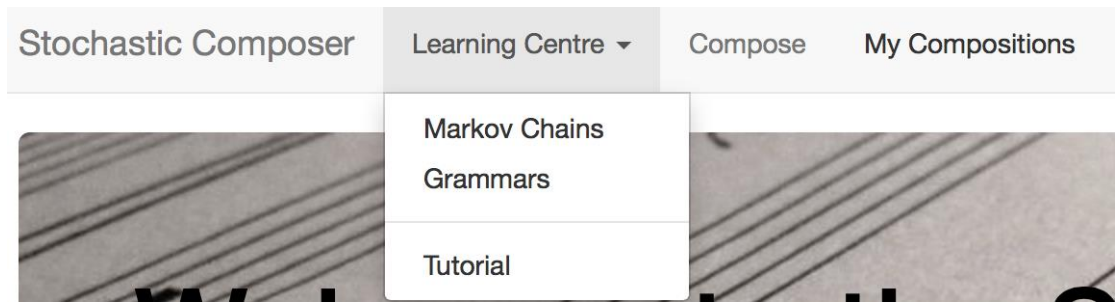


Figure 45: Learning Centre options

By clicking on the first two options one will be able to learn more about theoretical aspects of Markov Chains and Generative Grammars in music. The overview of Markov Chains covers the foundations of the concept, principal terms (like the order of a chain), and also provides linguistic and music examples. The overview of Generative Grammars includes essential definitions, which also cover stochastic and web grammars, together with relevant examples. The Tutorial part contains the following sections:

1. Tutorials:
  - Finale – Export XML
  - Sibelius – Export XML
  - Markov Chain – Compose
  - Generative Grammars – Compose
2. Examples:
  - Linear Grammar
  - Stochastic Grammar
  - Web Grammar
  - Using Markov Chains and Grammars together

The next tab – Compose – allows the user to navigate to composition sections for Markov Chains and Generative Grammars as well as to return to the Learning Centre (Figure 46):

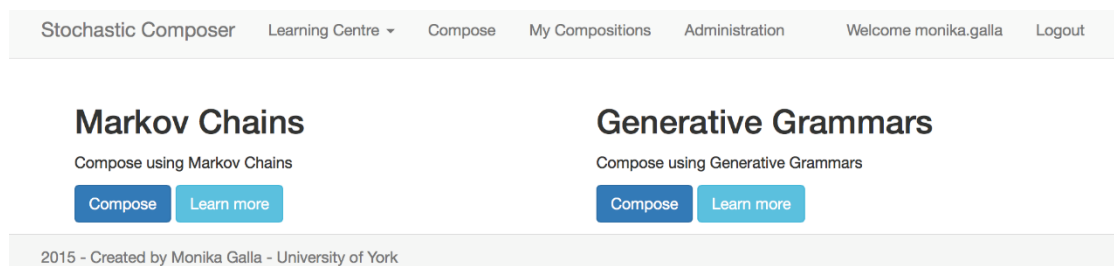


Figure 46: Compose section

My Compositions contains all the material created and uploaded by the user (Figure 47):

Stochastic Composer Learning Centre ▾ Compose My Compositions Administration Welcome administrator Logout

⊕ Upload Xml File (Composition)

**My Compositions**

Id	Title	Generation Type	Date Created	Description	Actions:
<input type="text" value="search"/>	<input type="text" value="search"/>	<input type="text" value="search"/>	<input type="text" value="search"/>	<input type="text" value="search"/>	
7	mch	MarkovChain	2015-10-05 16:20:39	mch1	<a href="#">Download</a> <a href="#">Details</a> <a href="#">Edit</a> <a href="#">Delete</a>
8	mch	MarkovChain	2015-10-05 16:20:39	mch1	<a href="#">Download</a> <a href="#">Details</a> <a href="#">Edit</a> <a href="#">Delete</a>
10	grammars	LinearGrammar	2015-10-05 16:25:02	grammars	<a href="#">Download</a> <a href="#">Details</a> <a href="#">Edit</a> <a href="#">Delete</a>
12	mch	Uploaded	2015-10-11 14:41:06		<a href="#">Download</a> <a href="#">Details</a> <a href="#">Edit</a> <a href="#">Delete</a>
13	mch2	Uploaded	2015-10-11 14:41:06		<a href="#">Download</a> <a href="#">Details</a> <a href="#">Edit</a> <a href="#">Delete</a>

2015 - Created by Monika Galla - University of York

Figure 47: My Compositions

Every composition has a unique ID, title, and created date. The column ‘Generation type’ specifies if the composition was created with Markov Chains or Grammars, or was a simple upload. On the right-hand side a list of possible actions is available:

- Download – to save composition in XML format
- Details – to view composition. Clicking on this option allows the user to make changes to the values previously provided (Figure 48)
- Edit – to change the name
- Delete – to remove the composition from the database

**Composition Details:**

**Id** 18  
**Title** mch3  
**Description** mch3  
**Date Created** 2015-10-22 20:10:14  
**Generation Type** Markov Chain

[Download](#) [Compose](#) [Edit](#) [Delete](#)



Figure 48: Composition Details

Clicking on ‘Details’ presents the details of the composition (Figure 49):

Composition Details:

**Id** 21  
**Title** First Web  
**Description**  
**Date Created** 2015-10-25 08:49:51  
**Generation Type** Web Grammar

[Compose](#) [Edit](#) [Delete](#)

---

Details

Grammar Type: **Web**  
Start Symbol: **A#**  
Terminal Symbols: **b c d a e g**  
Non Terminal Symbols: **C B A A#**

Production Rules

Type	Left Side	Right Side	Conditional	Probability
Stochastic	A#	C B A C		1
Web	C	b	c d	1
Stochastic	A	a		1
Stochastic	B	c e g		0.5
Stochastic	B	a b b		0.5

Grammar Run	State/Melody
0	A#()
1	C() B() A() C()
2	(() b5 )()() c3 d4 )() e3 e4 g4 a4 (() b4 )()() c4 d4 )()

2015 - Created by Monika Galla - University of York

Figure 49: Details of the composition

By clicking the blue Compose button it is possible to make use of an existing composition for further processing.

### 3.3.4. Markov Chains

Markov chains allow the user to upload an existing melody from an XML file or to create a new melody in the software itself. The following elements can be selected:

- Pitch, with alterations
- Octave
- Length

It is also possible to insert rests by checking the IsRest box. An example of a melody created in the software can be found in Figure 50.

Markov Chain - Create Melody - Control Panel

[Upload Melody From Xml](#) [Generate Chain](#)

Order:  
1

Notes

[Add Note](#) [Remove All](#)

Pitch (A to G)	Octave (1 to 8)	Alter	Type	IsRest	Dots	Actions
E	4		quarter	<input type="checkbox"/>		<a href="#">Duplicate</a> <a href="#">Edit</a> <a href="#">Remove</a>
B	4		quarter	<input type="checkbox"/>		<a href="#">Duplicate</a> <a href="#">Edit</a> <a href="#">Remove</a>
G	4		quarter	<input type="checkbox"/>		<a href="#">Duplicate</a> <a href="#">Edit</a> <a href="#">Remove</a>
A	4		quarter	<input type="checkbox"/>		<a href="#">Duplicate</a> <a href="#">Edit</a> <a href="#">Remove</a>
G	4		quarter	<input type="checkbox"/>		<a href="#">Duplicate</a> <a href="#">Edit</a> <a href="#">Remove</a>
F	4		quarter	<input type="checkbox"/>		<a href="#">Duplicate</a> <a href="#">Edit</a> <a href="#">Remove</a>
A	4		quarter	<input type="checkbox"/>		<a href="#">Duplicate</a> <a href="#">Edit</a> <a href="#">Remove</a>

Figure 50: Melody created in Stochastic Composer

After creating or uploading a melody it is possible to generate a chain by simply clicking on the green Generate Chain button. All the nodes and transition probabilities are presented in a tabular form, as seen in Figure 51 .

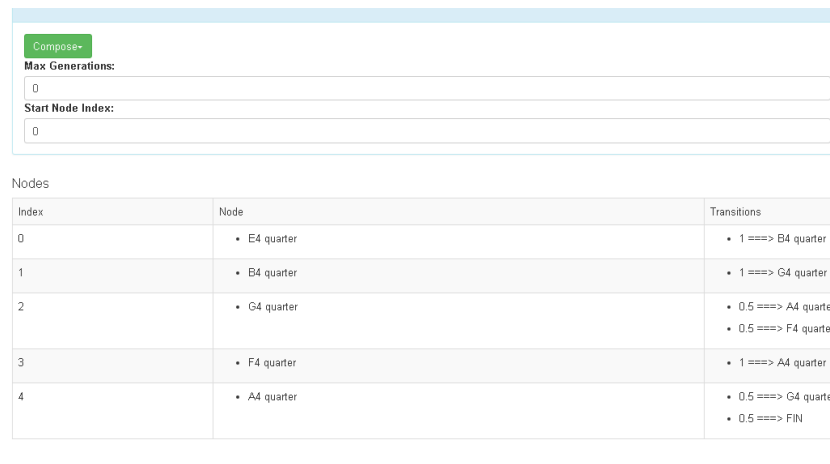


Figure 51: Markov chain details

By clicking on the Compose button one can select from two options:

- Simulate new melody
- Predict next notes

Before simulating a new melody it is necessary to select a maximum number of generations and the starting node. This is to avoid an infinite loop. After the output is generated it is possible to save it to the database and/or to download it in XML format, as shown in Figure 52.

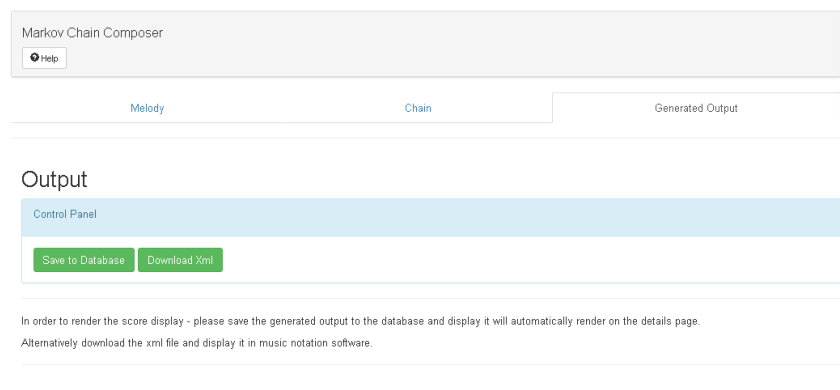


Figure 52: Markov chain generated output

When choosing to Save to Database, it is necessary to add the Title of the composition and an optional description, as presented in Figure 53.



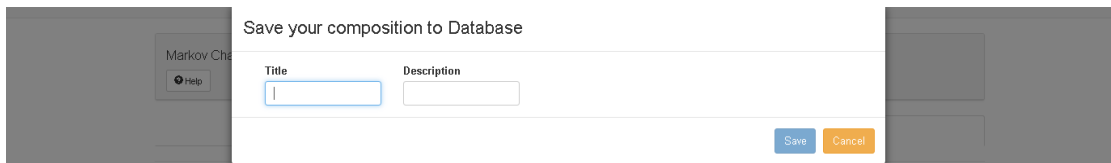


Figure 53: Saving output in a database

By navigating next to the Compositions section one can view and edit material already created by selecting Details next to the chosen composition (Figure 54).

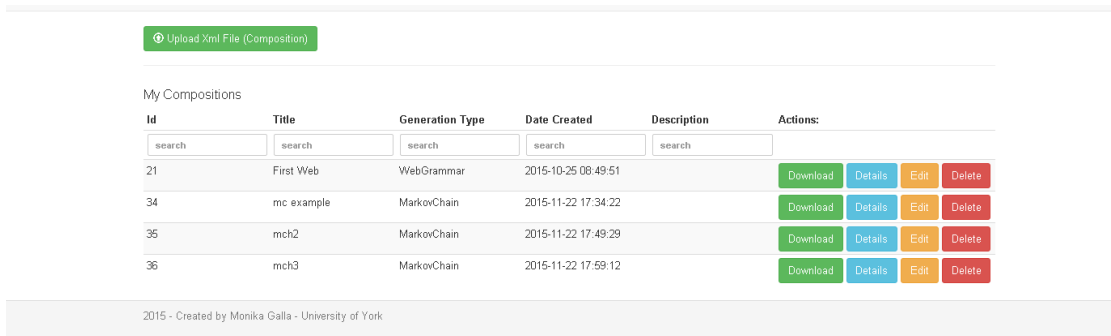


Figure 54: My compositions section

The Details of the composition include the generated outcome presented in staff notation, as well as a table with transitions and nodes (Figure 55).

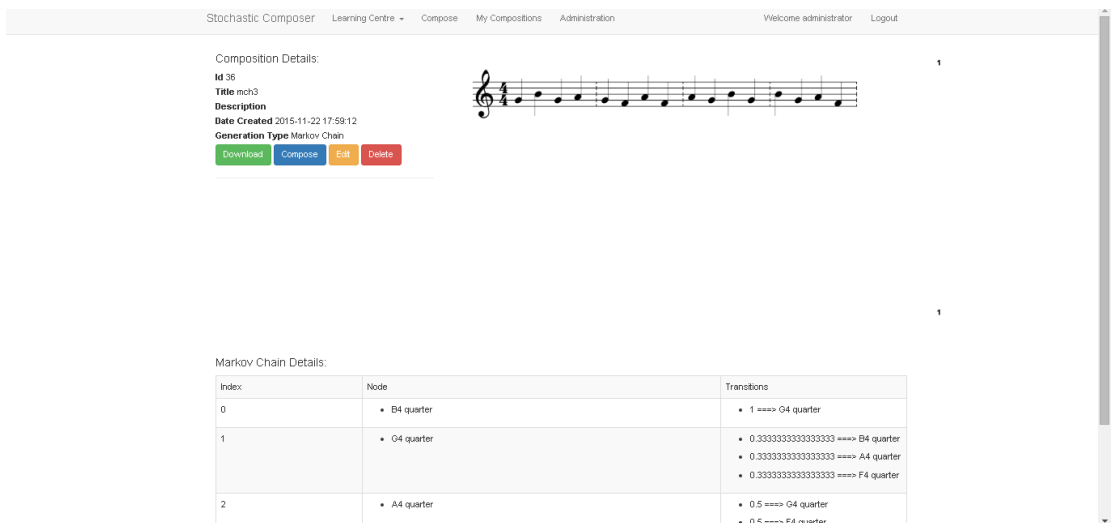


Figure 55: Details of Markov chain composition

By clicking on the Compose button one can return to the original input to modify it.

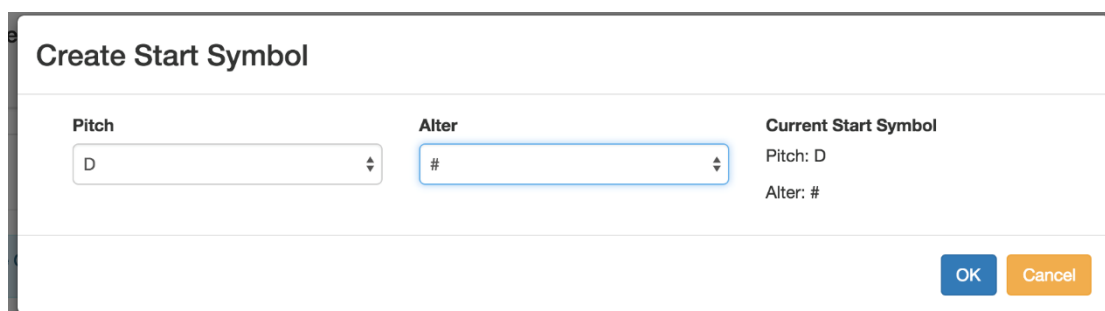
### 3.3.5. Generative Grammars

With the Stochastic Composer software it is possible to create one of three types of grammars:

- Linear Grammar
- Stochastic Grammar
- Web Grammar

#### **Linear Grammar**

When Linear Grammar is selected one must first decide on a starting symbol (Figure 56). It is necessary to provide the pitch and any needed alterations.



The screenshot shows a dialog box titled "Create Start Symbol". It features two dropdown menus: "Pitch" with "D" selected and "Alter" with "#" selected. To the right, under "Current Start Symbol", it displays "Pitch: D" and "Alter: #". At the bottom right are "OK" and "Cancel" buttons.

Figure 56: Start Symbol

The next step is to create production rules (Figure 57). It is necessary to include at list one production rule that includes the Start Symbol; otherwise the grammar would not be valid and the program will return an error. The program uses a type-3 right linear grammar, which means that it is not possible to create a rule with non-terminal symbols after terminals (for example,  $C \rightarrow CgC$ ).

## Create Production Rule

**Type**

Linear

---

**Left Side**

**Pitch**  **Alter**

---

**Right Side** Add Note

Pitch (A to G)	Alter	Actions
<input type="text" value="c"/>	<input type="text"/>	<span style="background-color: #e67e22; color: white; padding: 2px 5px; border-radius: 3px;">Remove</span>
<input type="text" value="C"/>	<input type="text"/>	<span style="background-color: #e67e22; color: white; padding: 2px 5px; border-radius: 3px;">Remove</span>

OK
Cancel

Figure 57: Adding new production rule

The next step is to validate the grammar. If all the production rules have been constructed correctly and at least one production rule contains the start symbol, the validation is successful and the Analyse Grammar section is presented (Figure 58). On the left hand side, in the Details section, a list of terminal and non-terminal symbols is displayed, together with the start symbol and grammar type. In the main section a summary of the rules is presented. The rules can be changed by clicking on the Create Grammar tab. In the Analyse Grammar section one also sets the maximum number of generations and selects the octave on which the Gaussian Kernel will be focused.

Create Grammar
Analyse Grammar
Generated Output

Generative Grammar - Grammar Verified!

Compose

**Max Generations (1 to MAX):**

**Select on which Octave Gaussian Kernel is focused (1 to 8)**

**Details**

Grammar Type: Linear

Start Symbol: D#

Terminal Symbols:

d# c

Non Terminal Symbols

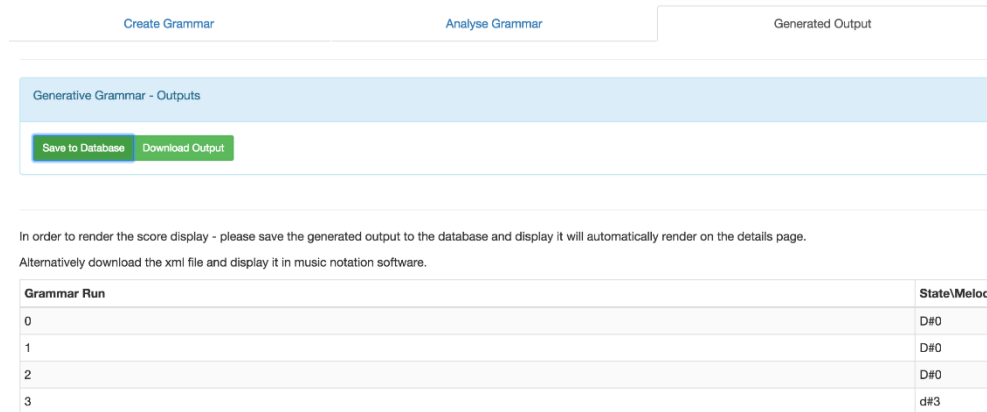
D# C

**Production Rules**

Type	Left Side	Right Side
Linear	D#	D#
Linear	D#	d#
Linear	D#	C
Linear	C	c
Linear	C	c C

Figure 58: Successful validation

To finally generate the output one needs to hit the Compose button. The outcome of each generation will be then displayed (Figure 59). At this stage it is possible to download the result or to save it to the database. After saving a result to the database one can still return to the grammar itself to modify it.



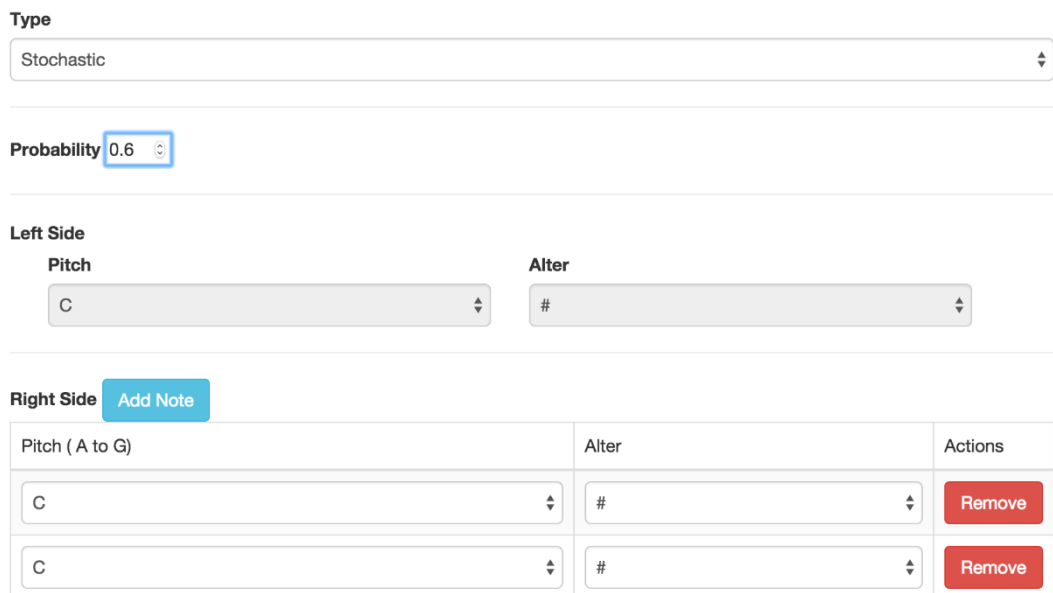
Grammar Run	State\Melody
0	D#0
1	D#0
2	D#0
3	d#3

Figure 59: Generated Output

### Stochastic Grammar

The major difference between a stochastic and a linear grammar is the inclusion of a probability for production rules that have the same left side (Figure 60).

#### Create Production Rule



Pitch (A to G)	Alter	Actions
C	#	Remove
C	#	Remove

Figure 60: Stochastic Production Rule

It is important to remember that the probability must equal 1; otherwise it will not be possible to validate the grammar. On Figure 61 the probabilities for symbol B sum to 0.9; therefore this grammar will not be validated.

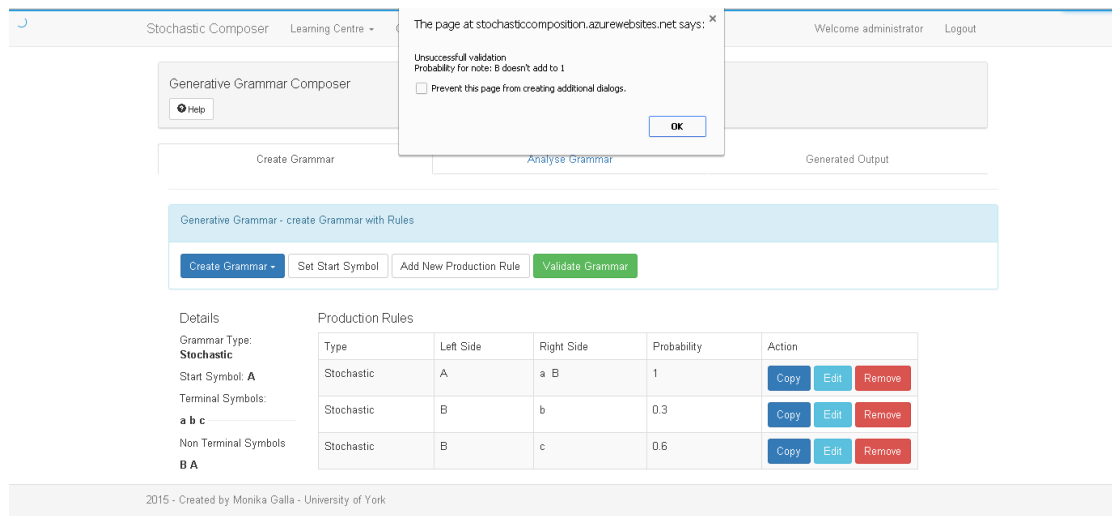


Figure 61: Probability of the rules are not equal to 1

Of course it is possible to combine linear and stochastic grammars. If there is only one possible rule for a particular non-terminal symbol, the probability of this rule should be set to 1, making it equivalent to a linear production rule.

### Web grammars

Web grammars allow the user to create not only horizontal but also vertical relationships with the symbols. It is possible to create a web grammar that also uses stochastic production rules. To define the type of rule desired, one simply chooses the relevant option from the drop down list, as presented in Figure 62.

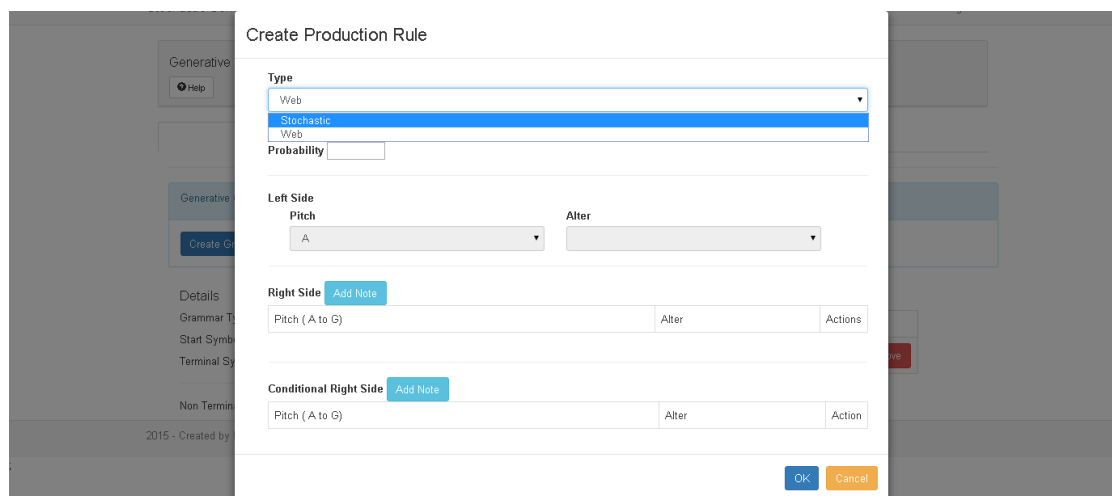


Figure 62: Production rules in web grammars

It is also possible to apply probabilities to the set of rules. Thus it is possible to create a grammar in which production rules can be in one or more dimensions, depending on the applied probability. This situation is illustrated in Figure 63.

Stochastic Composer   Learning Centre   Compose   My Compositions   Administration   Welcome administrator   Logout

---

Generative Grammar Composer

Help

Create Grammar   Analyse Grammar   Generated Output

---

Generative Grammar - create Grammar with Rules

Create Grammar   Set Start Symbol   Add New Production Rule   Validate Grammar

Details

Grammar Type: **Web**

Start Symbol: **A**

Terminal Symbols:

Non Terminal Symbols:

Production Rules

Type	Left Side	Right Side	Conditional	Probability	Action
Stochastic	A	a		0.1	Copy Edit Remove
Web	A	a	c	0.9	Copy Edit Remove

2015 - Created by Monika Galla - University of York

**Figure 63: Combined stochastic and web production rules**

### 3.4. Results

Five young composers were invited to test the Stochastic Composer software. The invitation resulted in 130 samples produced by the application. Overall, the Markov chain was significantly more popular, generating 102 samples. The whole experiment was conducted over the course of a month, during which minor bugs reported by the participants were fixed. During the registration process each of the participants was required to agree that the material produced would be analysed and presented in this thesis. The application is still available to anyone interested.

#### 3.4.1. Markov Chains

As mentioned before, Markov chains were used significantly more often than Generative Grammars. A possible reason is that, while chains do allow a user to generate and analyse music material, they are less complex than generative grammars. There are two methods of adding new material, and both were used: adding notes manually notes and uploading xml files.

#### Example 1



Figure 64: Stochastic Composer - Example MC1

Table 9: Stochastic Composer - Example MC1

Index	Node	Transitions
0	• C4 quarter Dots: 1	• 1 ==> E4 ♭ eighth
1	• E4 ♭ eighth	• 0.5 ==> C4 quarter Dots: 1 • 0.5 ==> F5 16th
2	• F5 16th	• 1 ==> E4 ♭ 16th
3	• E4 ♭ 16th	• 1 ==> G5 16th
4	• G5 16th	• 1 ==> F4 32nd
5	• F4 32nd	• 1 ==> A5 32nd
6	• A5 32nd	• 1 ==> C5 eighth
7	• C5 half Dots: 1	• 1 ==> C5 eighth
8	• C5 eighth	• 0.5 ==> C5 half Dots: 1 • 0.5 ==> REST eighth
9	• REST eighth	• 1 ==> FIN

The first example, presented in Figure 64 and Table 9, uses a first order Markov chain. In this simple example nine different nodes were generated. Most of them have

only one possible transition. Two nodes – E4  $\flat$  eighth and C5 eighth – have two possible transitions: for E4  $\flat$  , ‘C4 quarter Dots: 1’ and ‘F5 16<sup>th</sup>’; and for C5 eighth, ‘C5 dotted half’ and ‘REST eighth’. The probability of each alternative was set to 0.5. It is possible to enter a loop between the 7<sup>th</sup> and 8<sup>th</sup> note, as shown in Figure 65.



Figure 65: Stochastic Composer - Example MC1 - loop

The generations are moving between ‘C half Dot’ and ‘C eighth’. After a C eighth note, a C dotted half note occurs three times, while eighth note rest appears only once. It is important to note that even if the probability of each transition is equal to 0.5, in such a small sample this probability may not be reflected. It can be expected that the bigger the produced outcome is, the better will be the reflection of the probabilities of transitions.

**Example 2**



Figure 66: Stochastic Composer –Example - MC2

In a second example the presented outcome (Figure 66) contains only two different pitches, but some interesting relationships can be found in the transition diagram (Table 10).

Table 10: Stochastic Composer - Example - MC2

Index	Node	Transitions
0	<ul style="list-style-type: none"> <li>• C4 quarter</li> <li>• A4 quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; A4 quarter</li> </ul>
1	<ul style="list-style-type: none"> <li>• A4 quarter</li> <li>• A4 quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 0.6666666666666666 ==&gt; C4 quarter</li> <li>• 0.3333333333333333 ==&gt; A4 quarter</li> </ul>
2	<ul style="list-style-type: none"> <li>• A4 quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 0.5 ==&gt; A4 quarter</li> </ul>



- C4 quarter
- 0.5 ==> FIN

In this second-order Markov chain it is possible to distinguish three possible nodes that consist only of quarters A4 and C4. The generated material was started from node no 1. One might wonder why the piece ended on A4, since the transition diagram includes an ending on C4 with probability equal 0.5. The given ending results from the initial set up, in which the maximum generations was set to be 10. Therefore the process ended after ten generations, regardless of the last transition.

**Example 3**

A third example was created by third-order Markov chains (Figure 67 and Table 11):



Figure 67: Stochastic Composer –Example - MC3

Table 11: Stochastic Composer –Example - MC3

Index	Node	Transitions
0	<ul style="list-style-type: none"> <li>• D5 quarter</li> <li>• E5 eighth</li> <li>• D5 quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; D5 quarter</li> </ul>
1	<ul style="list-style-type: none"> <li>• E5 eighth</li> <li>• D5 quarter</li> <li>• D5 quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; B4 16th</li> </ul>
2	<ul style="list-style-type: none"> <li>• D5 quarter</li> <li>• D5 quarter</li> <li>• B4 16th</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; F4 16th</li> </ul>
3	<ul style="list-style-type: none"> <li>• D5 quarter</li> <li>• B4 16th</li> <li>• F4 16th</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; D4 eighth</li> </ul>
4	<ul style="list-style-type: none"> <li>• B4 16th</li> <li>• F4 16th</li> <li>• D4 eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; D4 eighth</li> </ul>
5	<ul style="list-style-type: none"> <li>• F4 16th</li> <li>• D4 eighth</li> <li>• D4 eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; A4 eighth</li> </ul>
6	<ul style="list-style-type: none"> <li>• D4 eighth</li> <li>• D4 eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; E4 eighth</li> </ul>

	<ul style="list-style-type: none"> <li>• A4 eighth</li> </ul>	
7	<ul style="list-style-type: none"> <li>• D4 eighth</li> <li>• A4 eighth</li> <li>• E4 eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; A4 eighth</li> </ul>
8	<ul style="list-style-type: none"> <li>• A4 eighth</li> <li>• E4 eighth</li> <li>• A4 eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; B4 eighth</li> </ul>
9	<ul style="list-style-type: none"> <li>• E4 eighth</li> <li>• A4 eighth</li> <li>• B4 eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; D4 eighth</li> </ul>
10	<ul style="list-style-type: none"> <li>• A4 eighth</li> <li>• B4 eighth</li> <li>• D4 eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; G4 quarter</li> </ul>
11	<ul style="list-style-type: none"> <li>• B4 eighth</li> <li>• D4 eighth</li> <li>• G4 quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; REST quarter</li> </ul>
12	<ul style="list-style-type: none"> <li>• D4 eighth</li> <li>• G4 quarter</li> <li>• REST quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; D6 eighth</li> </ul>
13	<ul style="list-style-type: none"> <li>• G4 quarter</li> <li>• REST quarter</li> <li>• D6 eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; D6 eighth</li> </ul>
14	<ul style="list-style-type: none"> <li>• REST quarter</li> <li>• D6 eighth</li> <li>• D6 eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; B4 quarter</li> </ul>
15	<ul style="list-style-type: none"> <li>• D6 eighth</li> <li>• D6 eighth</li> <li>• B4 quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; REST eighth</li> </ul>
16	<ul style="list-style-type: none"> <li>• D6 eighth</li> <li>• B4 quarter</li> <li>• REST eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; F4 16th</li> </ul>
17	<ul style="list-style-type: none"> <li>• B4 quarter</li> <li>• REST eighth</li> <li>• F4 16th</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; REST half Dots: 2</li> </ul>
18	<ul style="list-style-type: none"> <li>• REST eighth</li> <li>• F4 16th</li> <li>• REST half Dots: 2</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; G4 16th</li> </ul>
19	<ul style="list-style-type: none"> <li>• F4 16th</li> <li>• REST half Dots: 2</li> <li>• G4 16th</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; A4 16th</li> </ul>
20	<ul style="list-style-type: none"> <li>• REST half Dots: 2</li> <li>• G4 16th</li> <li>• A4 16th</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; F4 16th</li> </ul>
21	<ul style="list-style-type: none"> <li>• G4 16th</li> <li>• A4 16th</li> <li>• F4 16th</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; G4 16th</li> </ul>
22	<ul style="list-style-type: none"> <li>• A4 16th</li> <li>• F4 16th</li> <li>• G4 16th</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; A4 16th</li> </ul>
23	<ul style="list-style-type: none"> <li>• F4 16th</li> <li>• G4 16th</li> <li>• A4 16th</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; B4 16th</li> </ul>
24	<ul style="list-style-type: none"> <li>• G4 16th</li> <li>• A4 16th</li> <li>• B4 16th</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; D5 quarter</li> </ul>
25	<ul style="list-style-type: none"> <li>• A4 16th</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; D5 16th</li> </ul>

	<ul style="list-style-type: none"> <li>• B4 16th</li> <li>• D5 quarter</li> </ul>	
26	<ul style="list-style-type: none"> <li>• B4 16th</li> <li>• D5 quarter</li> <li>• D5 16th</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; D5 eighth</li> </ul>
27	<ul style="list-style-type: none"> <li>• D5 quarter</li> <li>• D5 16th</li> <li>• D5 eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; D5 quarter</li> </ul>
28	<ul style="list-style-type: none"> <li>• D5 16th</li> <li>• D5 eighth</li> <li>• D5 quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; D5 16th</li> </ul>
29	<ul style="list-style-type: none"> <li>• D5 eighth</li> <li>• D5 quarter</li> <li>• D5 16th</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; G5 16th</li> </ul>
30	<ul style="list-style-type: none"> <li>• D5 quarter</li> <li>• D5 16th</li> <li>• G5 16th</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; E5 half</li> </ul>
31	<ul style="list-style-type: none"> <li>• D5 16th</li> <li>• G5 16th</li> <li>• E5 half</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; E5 eighth</li> </ul>
32	<ul style="list-style-type: none"> <li>• G5 16th</li> <li>• E5 half</li> <li>• E5 eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; E5 quarter</li> </ul>
33	<ul style="list-style-type: none"> <li>• E5 eighth</li> <li>• E5 half</li> <li>• E5 eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; E5 quarter</li> </ul>
34	<ul style="list-style-type: none"> <li>• E5 half</li> <li>• E5 eighth</li> <li>• E5 quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; E5 eighth</li> </ul>
35	<ul style="list-style-type: none"> <li>• E5 eighth</li> <li>• E5 quarter</li> <li>• E5 eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; E5 half</li> </ul>
36	<ul style="list-style-type: none"> <li>• E5 quarter</li> <li>• E5 eighth</li> <li>• E5 half</li> </ul>	<ul style="list-style-type: none"> <li>• 0.5 ==&gt; E5 eighth</li> <li>• 0.5 ==&gt; D5 eighth</li> </ul>
37	<ul style="list-style-type: none"> <li>• E5 eighth</li> <li>• E5 half</li> <li>• D5 eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; D5 quarter</li> </ul>
38	<ul style="list-style-type: none"> <li>• E5 half</li> <li>• D5 eighth</li> <li>• D5 quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; D5 eighth</li> </ul>
39	<ul style="list-style-type: none"> <li>• D5 eighth</li> <li>• D5 quarter</li> <li>• D5 eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; C5 half</li> </ul>
40	<ul style="list-style-type: none"> <li>• D5 quarter</li> <li>• D5 eighth</li> <li>• C5 half</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; B4 eighth</li> </ul>
41	<ul style="list-style-type: none"> <li>• D5 eighth</li> <li>• C5 half</li> <li>• B4 eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; B4 quarter</li> </ul>
42	<ul style="list-style-type: none"> <li>• C5 half</li> <li>• B4 eighth</li> <li>• B4 quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; B4 eighth</li> </ul>
43	<ul style="list-style-type: none"> <li>• B4 eighth</li> <li>• B4 quarter</li> <li>• B4 eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; A4 eighth</li> </ul>

44	<ul style="list-style-type: none"> <li>• B4 quarter</li> <li>• B4 eighth</li> <li>• A4 eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; A4 half Dots: 1</li> </ul>
45	<ul style="list-style-type: none"> <li>• B4 eighth</li> <li>• A4 eighth</li> <li>• A4 half Dots: 1</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; A4 eighth</li> </ul>
46	<ul style="list-style-type: none"> <li>• A4 half Dots: 1</li> <li>• A4 eighth</li> <li>• D5 eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; D5 half Dots: 1</li> </ul>
47	<ul style="list-style-type: none"> <li>• A4 eighth</li> <li>• D5 eighth</li> <li>• D5 half Dots: 1</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; D5 eighth</li> </ul>
48	<ul style="list-style-type: none"> <li>• D5 eighth</li> <li>• D5 half Dots: 1</li> <li>• D5 eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; A4 eighth</li> </ul>
49	<ul style="list-style-type: none"> <li>• D5 half Dots: 1</li> <li>• D5 eighth</li> <li>• A4 eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; A4 half Dots: 1</li> </ul>
50	<ul style="list-style-type: none"> <li>• D5 eighth</li> <li>• A4 eighth</li> <li>• A4 half Dots: 1</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; A4 eighth</li> </ul>
51	<ul style="list-style-type: none"> <li>• A4 eighth</li> <li>• A4 half Dots: 1</li> <li>• A4 eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 0.6666666666666666 ==&gt; D5 eighth</li> <li>• 0.3333333333333333 ==&gt; FIN</li> </ul>

This example consists of a Markov chain of the third order with 51 nodes. While this example seems to be more complex than the previous ones, it is worth noticing that only nodes 36 and 51 contain more than one possibility. Therefore it was expected that the outcome of this generation would be almost the same as the input.

#### **Example 4**

In example 4, the input material provided as a sample was exactly the same as in example 3. The main difference is the order of chain, which was 1. The outcome is presented in Figure 68 and Table 12:



Figure 68: Stochastic Composer - Example - MC 4

Table 12: Stochastic Composer - Example - MC 4 - Transition Diagram

Index	Node	Transitions
0	• E4 eighth	• 1 ==> A4 eighth
1	• D4 eighth	• 0.3333333333333333 ==> D4 eighth • 0.3333333333333333 ==> A4 eighth • 0.3333333333333333 ==> G4 quarter
2	• G4 quarter	• 1 ==> REST quarter
3	• REST quarter	• 1 ==> D6 eighth
4	• D6 eighth	• 0.5 ==> D6 eighth • 0.5 ==> B4 quarter
5	• REST eighth	• 1 ==> F4 16th
6	• REST half Dots: 2	• 1 ==> G4 16th
7	• F4 16th	• 0.3333333333333333 ==> D4 eighth • 0.3333333333333333 ==> REST half Dots: 2 • 0.3333333333333333 ==> G4 16th
8	• G4 16th	• 1 ==> A4 16th
9	• A4 16th	• 0.5 ==> F4 16th • 0.5 ==> B4 16th
10	• B4 16th	• 0.5 ==> F4 16th • 0.5 ==> D5 quarter
11	• D5 16th	• 0.5 ==> D5 eighth • 0.5 ==> G5 16th
12	• G5 16th	• 1 ==> E5 half
13	• E5 quarter	• 1 ==> E5 eighth
14	• E5 eighth	• 0.2 ==> D5 quarter • 0.4 ==> E5 quarter • 0.4 ==> E5 half
15	• E5 half	• 0.6666666666666666 ==> E5 eighth • 0.3333333333333333 ==> D5 eighth
16	• D5 quarter	• 0.1666666666666666 ==> E5 eighth • 0.1666666666666666 ==> D5 quarter • 0.1666666666666666 ==> B4 16th • 0.3333333333333333 ==> D5 16th • 0.1666666666666666 ==> D5 eighth
17	• C5 half	• 1 ==> B4 eighth
18	• B4 quarter	• 0.5 ==> REST eighth • 0.5 ==> B4 eighth
19	• B4 eighth	• 0.3333333333333333 ==> D4 eighth • 0.3333333333333333 ==> B4 quarter • 0.3333333333333333 ==> A4 eighth
20	• D5 half Dots: 1	• 1 ==> D5 eighth
21	• D5 eighth	• 0.2857142857142857 ==> D5 quarter • 0.14285714285714285 ==> C5 half • 0.2857142857142857 ==> D5 half Dots: 1 • 0.2857142857142857 ==> A4 eighth
22	• A4 half Dots: 1	• 1 ==> A4 eighth
23	• A4 eighth	• 0.125 ==> E4 eighth • 0.125 ==> B4 eighth • 0.375 ==> A4 half Dots: 1 • 0.25 ==> D5 eighth • 0.125 ==> FIN

As can be observed in transition diagram (Table 12), this chain allows much more differentiation, which results in a richer outcome. Examples 3 and 4 perfectly

illustrate the need to consider very carefully the order of a Markov chain. One needs to be aware of the extremely thin line that separates randomness from repetitiveness when generating material with a Markov chain.

**Example 5**

The last example was created with the XML upload shown in Figure 69.



Figure 69: Stochastic Composer - Example - MC 5 Input

With this input three samples were created that differ only in the order of the Markov chain and the starting node.

The first sample started from node 0 and utilised a first-order Markov chain (Table 13 and Figure 70).



Figure 70: Stochastic Composer - Example - MC5a

Table 13: Stochastic Composer - Example - MC5a - Transition diagram

Index	Node	Transitions
0	• B3 eighth	• 1 ==> C4 eighth
1	• C4 eighth	• 0.5 ==> B3 eighth • 0.5 ==> E4 ♭ eighth
2	• E4 ♭ eighth	• 0.3333333333333333 ==> G4 quarter • 0.3333333333333333 ==> A4 quarter • 0.3333333333333333 ==> B4 ♭ quarter
3	• G4 quarter	• 0.5 ==> REST quarter • 0.5 ==> B4 ♭ quarter
4	• B4 ♭ quarter	• 1 ==> A4 quarter
5	• B4 ♭ eighth	• 0.6666666666666666 ==> A4 eighth • 0.3333333333333333 ==> A4 quarter

6	<ul style="list-style-type: none"> <li>A4 eighth</li> </ul>	<ul style="list-style-type: none"> <li>0.75 ==&gt; B4 b eighth</li> <li>0.25 ==&gt; G4 eighth</li> </ul>
7	<ul style="list-style-type: none"> <li>G4 eighth</li> </ul>	<ul style="list-style-type: none"> <li>1 ==&gt; F4# quarter</li> </ul>
8	<ul style="list-style-type: none"> <li>F4# quarter</li> </ul>	<ul style="list-style-type: none"> <li>1 ==&gt; REST quarter</li> </ul>
9	<ul style="list-style-type: none"> <li>E4 b quarter</li> </ul>	<ul style="list-style-type: none"> <li>1 ==&gt; D5 quarter</li> </ul>
10	<ul style="list-style-type: none"> <li>D5 quarter</li> </ul>	<ul style="list-style-type: none"> <li>1 ==&gt; F5# quarter</li> </ul>
11	<ul style="list-style-type: none"> <li>F5# quarter</li> </ul>	<ul style="list-style-type: none"> <li>1 ==&gt; A4 quarter</li> </ul>
12	<ul style="list-style-type: none"> <li>A4 quarter</li> </ul>	<ul style="list-style-type: none"> <li>0.6 ==&gt; REST quarter</li> <li>0.2 ==&gt; G4 quarter</li> <li>0.2 ==&gt; D4 quarter</li> </ul>
13	<ul style="list-style-type: none"> <li>D4 quarter</li> </ul>	<ul style="list-style-type: none"> <li>1 ==&gt; REST quarter</li> </ul>
14	<ul style="list-style-type: none"> <li>REST quarter</li> </ul>	<ul style="list-style-type: none"> <li>0.25 ==&gt; C4 eighth</li> <li>0.25 ==&gt; A4 eighth</li> <li>0.125 ==&gt; E4 b quarter</li> <li>0.25 ==&gt; REST quarter</li> <li>0.125 ==&gt; FIN</li> </ul>

In the output some randomness can be observed but there is a strong reflection of the half-tone motive. Also interesting appears to be the rhythmic domain, with some rhythmic shifts off the beat.

The second sample was based on a second-order Markov chain and started from node 4 (Figure 71 and Table 14):



Figure 71: Stochastic Composer –Example – MC5b

Table 14: Stochastic Composer –Example – MC5b – Transition diagram

Index	Node	Transitions
0	<ul style="list-style-type: none"> <li>E4 b eighth</li> <li>G4 quarter</li> </ul>	<ul style="list-style-type: none"> <li>1 ==&gt; REST quarter</li> </ul>
1	<ul style="list-style-type: none"> <li>G4 quarter</li> <li>REST quarter</li> </ul>	<ul style="list-style-type: none"> <li>1 ==&gt; C4 eighth</li> </ul>
2	<ul style="list-style-type: none"> <li>E4 b eighth</li> <li>A4 quarter</li> </ul>	<ul style="list-style-type: none"> <li>1 ==&gt; REST quarter</li> </ul>
3	<ul style="list-style-type: none"> <li>REST quarter</li> <li>C4 eighth</li> </ul>	<ul style="list-style-type: none"> <li>1 ==&gt; B3 eighth</li> </ul>
4	<ul style="list-style-type: none"> <li>C4 eighth</li> </ul>	<ul style="list-style-type: none"> <li>1 ==&gt; C4 eighth</li> </ul>

	<ul style="list-style-type: none"> <li>• B3 eighth</li> </ul>	
5	<ul style="list-style-type: none"> <li>• B3 eighth</li> <li>• C4 eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; E4 b eighth</li> </ul>
6	<ul style="list-style-type: none"> <li>• C4 eighth</li> <li>• E4 b eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 0.3333333333333333 ==&gt; G4 quarter</li> <li>• 0.3333333333333333 ==&gt; A4 quarter</li> <li>• 0.3333333333333333 ==&gt; B4 b quarter</li> </ul>
7	<ul style="list-style-type: none"> <li>• E4 b eighth</li> <li>• B4 b quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; A4 quarter</li> </ul>
8	<ul style="list-style-type: none"> <li>• A4 quarter</li> <li>• G4 quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; B4 b quarter</li> </ul>
9	<ul style="list-style-type: none"> <li>• G4 quarter</li> <li>• B4 b quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; A4 quarter</li> </ul>
10	<ul style="list-style-type: none"> <li>• B4 b quarter</li> <li>• A4 quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 0.5 ==&gt; G4 quarter</li> <li>• 0.5 ==&gt; REST quarter</li> </ul>
11	<ul style="list-style-type: none"> <li>• B4 b eighth</li> <li>• A4 quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; REST quarter</li> </ul>
12	<ul style="list-style-type: none"> <li>• A4 quarter</li> <li>• REST quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 0.3333333333333333 ==&gt; C4 eighth</li> <li>• 0.6666666666666666 ==&gt; A4 eighth</li> </ul>
13	<ul style="list-style-type: none"> <li>• REST quarter</li> <li>• A4 eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; B4 b eighth</li> </ul>
14	<ul style="list-style-type: none"> <li>• A4 eighth</li> <li>• B4 b eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 0.6666666666666666 ==&gt; A4 eighth</li> <li>• 0.3333333333333333 ==&gt; A4 quarter</li> </ul>
15	<ul style="list-style-type: none"> <li>• B4 b eighth</li> <li>• A4 eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 0.5 ==&gt; B4 b eighth</li> <li>• 0.5 ==&gt; G4 eighth</li> </ul>
16	<ul style="list-style-type: none"> <li>• A4 eighth</li> <li>• G4 eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; F4# quarter</li> </ul>
17	<ul style="list-style-type: none"> <li>• G4 eighth</li> <li>• F4# quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; REST quarter</li> </ul>
18	<ul style="list-style-type: none"> <li>• F4# quarter</li> <li>• REST quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; E4 b quarter</li> </ul>
19	<ul style="list-style-type: none"> <li>• REST quarter</li> <li>• E4 b quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; D5 quarter</li> </ul>
20	<ul style="list-style-type: none"> <li>• E4 b quarter</li> <li>• D5 quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; F5# quarter</li> </ul>
21	<ul style="list-style-type: none"> <li>• D5 quarter</li> <li>• F5# quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; A4 quarter</li> </ul>
22	<ul style="list-style-type: none"> <li>• F5# quarter</li> <li>• A4 quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; D4 quarter</li> </ul>
23	<ul style="list-style-type: none"> <li>• A4 quarter</li> <li>• D4 quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; REST quarter</li> </ul>
24	<ul style="list-style-type: none"> <li>• D4 quarter</li> <li>• REST quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; REST quarter</li> </ul>
25	<ul style="list-style-type: none"> <li>• REST quarter</li> <li>• REST quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 0.5 ==&gt; REST quarter</li> <li>• 0.5 ==&gt; FIN</li> </ul>

The resemblance to the original sample is more evident here. As can be seen on the transition diagram presented in Table 14 this chain still allows some randomness, but the result recalls the original input to a great extent.

The last sample was based on a third-order Markov chain and starts from node 0 (Table 15 and Figure 72).





Figure 72: Stochastic Composer –Example – MC5c

Table 15: Stochastic Composer –Example – MC5c – Transition diagram

Index	Node	Transitions
0	<ul style="list-style-type: none"> <li>• C4 eighth</li> <li>• E4 ♭ eighth</li> <li>• G4 quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; REST quarter</li> </ul>
1	<ul style="list-style-type: none"> <li>• E4 ♭ eighth</li> <li>• G4 quarter</li> <li>• REST quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; C4 eighth</li> </ul>
2	<ul style="list-style-type: none"> <li>• G4 quarter</li> <li>• REST quarter</li> <li>• C4 eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; B3 eighth</li> </ul>
3	<ul style="list-style-type: none"> <li>• C4 eighth</li> <li>• E4 ♭ eighth</li> <li>• A4 quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; REST quarter</li> </ul>
4	<ul style="list-style-type: none"> <li>• E4 ♭ eighth</li> <li>• A4 quarter</li> <li>• REST quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; C4 eighth</li> </ul>
5	<ul style="list-style-type: none"> <li>• A4 quarter</li> <li>• REST quarter</li> <li>• C4 eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; B3 eighth</li> </ul>
6	<ul style="list-style-type: none"> <li>• REST quarter</li> <li>• C4 eighth</li> <li>• B3 eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; C4 eighth</li> </ul>
7	<ul style="list-style-type: none"> <li>• C4 eighth</li> <li>• B3 eighth</li> <li>• C4 eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; E4 ♭ eighth</li> </ul>
8	<ul style="list-style-type: none"> <li>• B3 eighth</li> <li>• C4 eighth</li> <li>• E4 ♭ eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 0.3333333333333333 ==&gt; G4 quarter</li> <li>• 0.3333333333333333 ==&gt; A4 quarter</li> <li>• 0.3333333333333333 ==&gt; B4 ♭ quarter</li> </ul>
9	<ul style="list-style-type: none"> <li>• C4 eighth</li> <li>• E4 ♭ eighth</li> <li>• B4 ♭ quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; A4 quarter</li> </ul>
10	<ul style="list-style-type: none"> <li>• E4 ♭ eighth</li> <li>• B4 ♭ quarter</li> <li>• A4 quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; G4 quarter</li> </ul>
11	<ul style="list-style-type: none"> <li>• B4 ♭ quarter</li> <li>• A4 quarter</li> <li>• G4 quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; B4 ♭ quarter</li> </ul>
12	<ul style="list-style-type: none"> <li>• A4 quarter</li> <li>• G4 quarter</li> <li>• B4 ♭ quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; A4 quarter</li> </ul>
13	<ul style="list-style-type: none"> <li>• G4 quarter</li> <li>• B4 ♭ quarter</li> <li>• A4 quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 1 ==&gt; REST quarter</li> </ul>

14	<ul style="list-style-type: none"> <li>• B4 ♭ quarter</li> <li>• A4 quarter</li> <li>• REST quarter</li> </ul>	• 1 ==> A4 eighth
15	<ul style="list-style-type: none"> <li>• B4 ♭ eighth</li> <li>• A4 eighth</li> <li>• B4 ♭ eighth</li> </ul>	• 1 ==> A4 quarter
16	<ul style="list-style-type: none"> <li>• A4 eighth</li> <li>• B4 ♭ eighth</li> <li>• A4 quarter</li> </ul>	• 1 ==> REST quarter
17	<ul style="list-style-type: none"> <li>• B4 ♭ eighth</li> <li>• A4 quarter</li> <li>• REST quarter</li> </ul>	• 1 ==> A4 eighth
18	<ul style="list-style-type: none"> <li>• A4 quarter</li> <li>• REST quarter</li> <li>• A4 eighth</li> </ul>	• 1 ==> B4 ♭ eighth
19	<ul style="list-style-type: none"> <li>• REST quarter</li> <li>• A4 eighth</li> <li>• B4 ♭ eighth</li> </ul>	• 1 ==> A4 eighth
20	<ul style="list-style-type: none"> <li>• A4 eighth</li> <li>• B4 ♭ eighth</li> <li>• A4 eighth</li> </ul>	<ul style="list-style-type: none"> <li>• 0.5 ==&gt; B4 ♭ eighth</li> <li>• 0.5 ==&gt; G4 eighth</li> </ul>
21	<ul style="list-style-type: none"> <li>• B4 ♭ eighth</li> <li>• A4 eighth</li> <li>• G4 eighth</li> </ul>	• 1 ==> F4# quarter
22	<ul style="list-style-type: none"> <li>• A4 eighth</li> <li>• G4 eighth</li> <li>• F4# quarter</li> </ul>	• 1 ==> REST quarter
23	<ul style="list-style-type: none"> <li>• G4 eighth</li> <li>• F4# quarter</li> <li>• REST quarter</li> </ul>	• 1 ==> E4 ♭ quarter
24	<ul style="list-style-type: none"> <li>• F4# quarter</li> <li>• REST quarter</li> <li>• E4 ♭ quarter</li> </ul>	• 1 ==> D5 quarter
25	<ul style="list-style-type: none"> <li>• REST quarter</li> <li>• E4 ♭ quarter</li> <li>• D5 quarter</li> </ul>	• 1 ==> F5# quarter
26	<ul style="list-style-type: none"> <li>• E4 ♭ quarter</li> <li>• D5 quarter</li> <li>• F5# quarter</li> </ul>	• 1 ==> A4 quarter
27	<ul style="list-style-type: none"> <li>• D5 quarter</li> <li>• F5# quarter</li> <li>• A4 quarter</li> </ul>	• 1 ==> D4 quarter
28	<ul style="list-style-type: none"> <li>• F5# quarter</li> <li>• A4 quarter</li> <li>• D4 quarter</li> </ul>	• 1 ==> REST quarter
29	<ul style="list-style-type: none"> <li>• A4 quarter</li> <li>• D4 quarter</li> <li>• REST quarter</li> </ul>	• 1 ==> REST quarter
30	<ul style="list-style-type: none"> <li>• D4 quarter</li> <li>• REST quarter</li> <li>• REST quarter</li> </ul>	• 1 ==> REST quarter
31	<ul style="list-style-type: none"> <li>• REST quarter</li> <li>• REST quarter</li> <li>• REST quarter</li> </ul>	• 1 ==> FIN

It can be observed that the most interesting results were obtained when using the first-order Markov chain. While the organisation of the melody still recalls the original sample, the rhythm is much more interesting. With the increasing orders of the chain one notices that the outcome starts to resemble the original material, in both melodic and rhythmic aspects. In this example it is possible to note how important it is to consider an appropriate order for a Markov chain to achieve satisfactory results.

### 3.4.2. Generative Grammars

As stated before, generative grammars appeared to be less popular among the participants in the experiment. This is probably due to their complexity. To achieve satisfactory results the grammar needs to be carefully designed.

Overall 28 samples were created. Interestingly, some of the participants decided to combine grammars with Markov chains or Gaussian distribution on a larger scale that was imagined when making Stochastic Composer.

#### **Example 1**

In the first example a grammar was created from the motive BACH. The start symbol of this grammar is A#. It was chosen in order to distinguish the start symbol from the rest of non-terminal symbols, while the integrity of the grammar is still preserved, as the A# is enharmonically B b. (Please note that the Stochastic Composer uses English nomenclature, while the BACH motive was created in German note names; therefore BACH is expressed here as B b ACB, where B is replaced by B b and H is replaced by B.)

The grammar in question has the following structure:

GSC1 = (V<sub>N</sub>, V<sub>T</sub>, P, A#), where

V<sub>N</sub> = { B b, A, C, B, A# },

V<sub>T</sub> = { b b, a, c, b },

P :

Type	Left Side	Right Side	Probability
Stochastic	A#	B b A C B	1
Stochastic	A	A A	0.2
Stochastic	A	a	0.4
Stochastic	A	c B	0.2
Stochastic	A	C B	0.2

Stochastic	B	B B	0.2
Stochastic	B	b	0.4
Stochastic	B	b ♭ A	0.2
Stochastic	B	B ♭ A	0.2
Stochastic	C	C C	0.2
Stochastic	C	c	0.4
Stochastic	C	b B ♭	0.2
Stochastic	C	B B ♭	0.2
Stochastic	B ♭	B ♭ B ♭	0.2
Stochastic	B ♭	b ♭	0.4
Stochastic	B ♭	A C	0.2
Stochastic	B ♭	a C	0.2

Several generations were created with this grammar; some examples can be found in Figure 73 to Figure 77.



Figure 73: Stochastic Composer - GSC1 - Sample 1



Figure 74: Stochastic Composer - GSC1 - Sample 2



Figure 75: Stochastic Composer - GSC1 - Sample 3

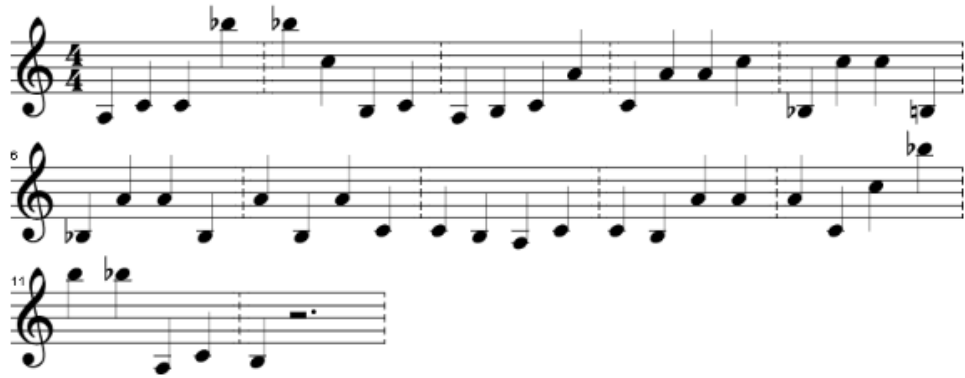


Figure 76: Stochastic Composer - GSC1 - Sample 4



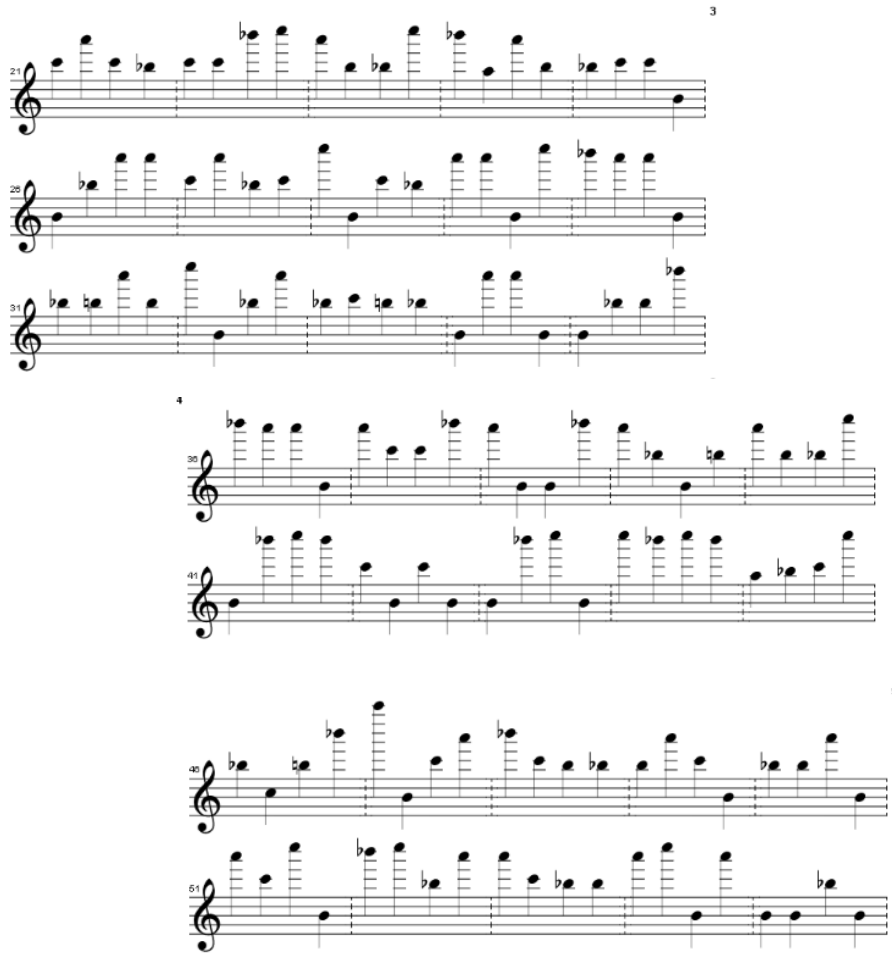


Figure 77: Stochastic Composer - GSC1 - Sample 5

Sample 1 was focused on the fourth octave, while samples 2, 3, 4 and 5 were focused on the fifth octave. The main difference is in the length of the samples. While the grammar remained the same, it can be easily noticed that the second sample is the shortest, with only 8 generations, while the first, third and fourth samples were created after 15, 17, and 25 generations respectively. This is due to the fact that even a probability of 0.4 can cause the grammar to quickly convert to terminal symbols. The most interesting is the last, fifth sample, where the grammar never terminated. The maximum number of generations was set to 30 and this is the point at which the outcome ended. It is important to notice that some of the symbols remained non-terminal; but due to the limitation of the maximum number of generations, these were converted automatically to the relevant terminal symbols:

---

<b>Generation</b>	<b>b ♭ 5 c7 b4 b ♭ 6 a6 b4 b4 a6 c6 a6 c6 b4 b4 b ♭ 5 b ♭ 5 a6 a6 a6 b5 a6 c6 c6 b4 b</b>
<b>30</b>	<b>♭ 6 c7 b ♭ 5 a6 b ♭ 5 b ♭ 5 a6 b5 a6 C5 b ♭ 5 c7 b ♭ 6 b ♭ 6 A7 a6 C5 b4 c6 b ♭</b>
	<b>6 A7 B ♭ 6 A5 b ♭ 5 a6 b4 b ♭ 5 c7 b4 b ♭ 5 a6 b4 b ♭ 5 b ♭ 5 a6 b5 b ♭ 5 b ♭</b>

---

---

5 a6 c6 b4 b b 5 b b 5 c6 b5 a6 b5 b b 5 b4 b b 5 b4 a6 b b 6 a6 a6 b5 b b  
 5 c6 a6 c6 b b 5 c6 c6 b b 6 c7 a6 B5 B b 5 c7 B b 6 A5 a6 b5 b b 5 c6 c6 b4 b4 b b  
 5 a6 a6 c6 a6 b b 5 c6 c7 b4 c6 b b 5 a6 a6 b4 c7 b b 6 a6 a6 b4 b b  
 5 b5 a6 b5 c7 b4 b b 5 a6 b b 5 c6 b5 b b 5 b4 a6 a6 b4 b4 b b 5 B b 5 B b 6 b b  
 6 a6 a6 b4 a6 c6 c6 b b 6 a6 b4 b4 b b 6 a6 b b 5 b4 b5 a6 b5 b b 5 c7 b4 b b 6 c7 b  
 b 6 c6 b4 c6 b4 b4 b b 6 c7 b4 c7 b b 6 c7 B b 6 A5 b b 5 c6 c7 b b 5 C5 B5 b b  
 6 A7 b4 c6 a6 b b 6 c6 b5 b b 5 b5 a6 c6 b4 b b 5 b b 5 a6 b4 a6 c6 c7 b4 b b 6 c7 b  
 b 5 a6 a6 c6 b b 5 b b 5 a6 c7 b4 a6 b4 b4 b b 5 b4

---

**Example 2**

The second example is based on the same terminal and non-terminal symbols, but it differs from example 1 in the values of the probabilities in the production rules. Overall the probabilities were designed to favour terminal symbols, so it was expected that the samples would be shorter than in the previous example.

$GSC2 = (V_N, V_T, P, A\#)$ , where

$V_N = \{B, A, C, H, A\# \}$ ,

$V_T = \{b, a, c, h \}$ ,

**P :**

Type	Left Side	Right Side	Probability
Stochastic	A#	B b A C B	1
Stochastic	A	A A	0.1
Stochastic	A	a	0.7
Stochastic	A	c B	0.1
Stochastic	A	C B	0.1
Stochastic	B	B B	0.1
Stochastic	B	b	0.7
Stochastic	B	b b A	0.1
Stochastic	B	B b A	0.1
Stochastic	C	C C	0.1
Stochastic	C	c	0.7
Stochastic	C	b B b	0.1
Stochastic	C	B B b	0.1
Stochastic	B b	B b B b	0.1
Stochastic	B b	b b	0.7
Stochastic	B b	A C	0.1

---

<b>Stochastic</b>	B ♭	a C	0.1
-------------------	-----	-----	-----

This grammar was run twice. The first run, after six generations (Table 16), led to a two-bar sample (Figure 78):



Figure 78: Stochastic Composer - GSC2 - Sample 1

Table 16: Stochastic Composer - GSC2 - Sample 1 - Generations

Grammar Run	State\Melody
0	A#0
1	B ♭ 0 A0 C0 B0
2	b ♭ 6 a6 b5 B ♭ 5 b4
3	b ♭ 6 a6 b5 A3 C4 b4
4	b ♭ 6 a6 b5 a6 C2 C5 b4
5	b ♭ 6 a6 b5 a6 b5 B ♭ 5 c4 b4
6	b ♭ 6 a6 b5 a6 b5 b ♭ 6 c4 b4

The second run produced the original BACH motive (Figure 79):



Figure 79: Stochastic Composer - GSC2 - Sample 2

This outcome was achieved after just two generations (Table 17). This is an excellent example to show how important it is to carefully design production rules and probabilities to achieve desired results.

Table 17: Stochastic Composer - GSC2 - Sample 2 - Generations

Grammar Run	State\Melody
0	A#0
1	B ♭ 0 A0 C0 B0
2	b ♭ 3 a5 c5 b4

**Example 3**

The last example applied, to a large extent, Gaussian distributions. The sets of terminal and non-terminal symbols compromise only 7 symbols – notes without any alterations. The interesting aspects of this grammar are the production rules, which favour the closest possible notes with the relevant probabilities:

GSC3 ( $V_N, V_T, P, A$ ), where



$V_n = \{a, b, c, d, e, f, g\}$ ,

$V_T = \{A, B, C, D, E, F, G\}$ ,

P :

Type	Left Side	Right Side	Probability
Stochastic	A	a	0.382925
Stochastic	A	B C	0.24173
Stochastic	A	C E	0.060598
Stochastic	A	D G	0.0062095
Stochastic	A	E B	0.0062095
Stochastic	A	F D	0.060598
Stochastic	A	G F	0.24173
Stochastic	B	b	0.382925
Stochastic	B	C D	0.24173
Stochastic	B	D F	0.060598
Stochastic	B	E A	0.0062095
Stochastic	B	F C	0.0062095
Stochastic	B	G E	0.060598
Stochastic	B	G A	0.24173
Stochastic	C	c	0.382925
Stochastic	C	D E	0.24173
Stochastic	C	E G	0.060598
Stochastic	C	F B	0.0062095
Stochastic	C	G D	0.0062095
Stochastic	C	A F	0.060598
Stochastic	C	B A	0.24173
Stochastic	D	d	0.382925
Stochastic	D	E F	0.24173
Stochastic	D	F A	0.060598
Stochastic	D	G C	0.0062095
Stochastic	D	A E	0.0062095
Stochastic	D	B G	0.060598
Stochastic	D	C B	0.24173
Stochastic	E	e	0.382925
Stochastic	E	F G	0.24173
Stochastic	E	G B	0.060598
Stochastic	E	A D	0.0062095
Stochastic	E	B F	0.0062095
Stochastic	E	C A	0.060598
Stochastic	E	D C	0.24173
Stochastic	F	f	0.382925
Stochastic	F	G A	0.24173
Stochastic	F	A C	0.060598
Stochastic	F	B E	0.0062095
Stochastic	F	C G	0.0062095
Stochastic	F	D B	0.060598
Stochastic	F	E D	0.24173
Stochastic	G	g	0.382925
Stochastic	G	A B	0.24173
Stochastic	G	B D	0.060598
Stochastic	G	C F	0.0062095
Stochastic	G	D A	0.0062095
Stochastic	G	E C	0.060598
Stochastic	G	F E	0.24173

As can be observed, the highest probabilities always generate a terminal symbol from a non-terminal symbol, where both terminal and non-terminal symbol are represented by the same letter. The next most likely favour the closest notes. For example, from A it is most probable to generate a. The next most likely outcome is BC or GF – the notes that are closest to A. The third most likely results in FD and CE, and the least probable are EB and DG, since E and D are the furthest notes possible in the grammar. In addition the distance between possible notes increases with decreased probability. The probability distribution (an approximation) is shown in Figure 80.

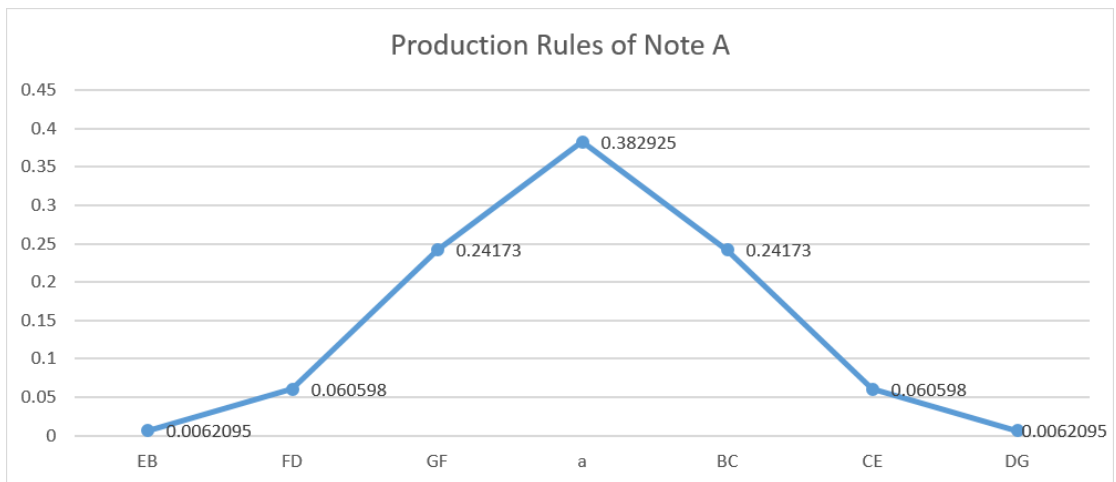


Figure 80: Stochastic Composer - Production Rule Probability Distribution for Note 'A'

From this grammar two samples were produced, presented in Figure 81 and Figure 82.



3

Two staves of musical notation. The top staff is in treble clef and the bottom staff is in bass clef. Both staves contain eighth and sixteenth notes. A measure number '3' is positioned above the top staff.

Three staves of musical notation. The top staff is in treble clef and the two bottom staves are in bass clef. The notation continues with eighth and sixteenth notes.

Two staves of musical notation. The top staff is in treble clef and the bottom staff is in bass clef. The notation continues with eighth and sixteenth notes.

Two staves of musical notation. The top staff is in treble clef and the bottom staff is in bass clef. The notation continues with eighth and sixteenth notes.

Two staves of musical notation. The top staff is in treble clef and the bottom staff is in bass clef. The notation continues with eighth and sixteenth notes.

Two staves of musical notation. The top staff is in treble clef and the bottom staff is in bass clef. The notation continues with eighth and sixteenth notes.



Figure 81: Stochastic Composer - GSC3 - Sample 1



Figure 82: Stochastic Composer - GSC3 - Sample 2

From the first sample, which is significantly longer and more complicated, a Markov chain was created. The next step was to create a new melody based on this chain (Figure 83):



Figure 83: Stochastic Composer - GSC - Markov chain from generative grammar

The transition diagram can be found in Table 18:

Table 18: Stochastic Composer - GSC3 - Markov Chain from generative grammar - Transition diagram

Index	Node	Transitions
0	• A2 quarter	• 1 ==> F3 quarter
1	• F3 quarter	• 1 ==> E5 quarter
2	• C5 quarter	• 1 ==> A4 quarter
3	• E6 quarter	• 0.3333333333333333 ==> E5 quarter • 0.3333333333333333 ==> C3 quarter • 0.3333333333333333 ==> G1 quarter
4	• G3 quarter	• 1 ==> C6 quarter
5	• C6 quarter	• 1 ==> G6 quarter
6	• B6 quarter	• 1 ==> E5 quarter
7	• E4 quarter	• 1 ==> F5 quarter
8	• C2 quarter	• 1 ==> D4 quarter
9	• G5 quarter	• 1 ==> B4 quarter
10	• D4 quarter	• 0.4 ==> C4 quarter • 0.0666666666666667 ==> C3 quarter • 0.1333333333333333 ==> E6 quarter • 0.0666666666666667 ==> F4 quarter • 0.0666666666666667 ==> E5 quarter • 0.0666666666666667 ==> A5 quarter • 0.0666666666666667 ==> B6 quarter • 0.0666666666666667 ==> G1 quarter • 0.0666666666666667 ==> B4 quarter
11	• G4 quarter	• 0.15 ==> E5 quarter • 0.45 ==> A5 quarter • 0.1 ==> C7 quarter • 0.05 ==> F4 quarter • 0.2 ==> D5 quarter • 0.05 ==> E3 quarter
12	• E3 quarter	• 0.2222222222222222 ==> A3 quarter • 0.1111111111111111 ==> G4 quarter • 0.2222222222222222 ==> E5 quarter • 0.1111111111111111 ==> C2 quarter • 0.1111111111111111 ==> G1 quarter • 0.1111111111111111 ==> F4 quarter • 0.1111111111111111 ==> G6 quarter
13	• G1 quarter	• 1 ==> F5 quarter
14	• F5 quarter	• 0.4444444444444444 ==> E3 quarter • 0.0555555555555555 ==> C3 quarter • 0.0555555555555555 ==> G6 quarter • 0.0555555555555555 ==> D4 quarter • 0.0555555555555555 ==> B5 quarter • 0.0555555555555555 ==> A3 quarter • 0.0555555555555555 ==> B4 quarter • 0.1111111111111111 ==> F4 quarter • 0.0555555555555555 ==> D5 quarter • 0.0555555555555555 ==> E5 quarter
15	• C7 quarter	• 1 ==> B3 quarter
16	• B3 quarter	• 0.4166666666666667 ==> C3 quarter • 0.25 ==> D5 quarter • 0.0833333333333333 ==> G6 quarter • 0.1666666666666666 ==> B4 quarter • 0.0833333333333333 ==> E5 quarter
17	• B4 quarter	• 0.1333333333333333 ==> G4 quarter • 0.5333333333333333 ==> A3 quarter



		<ul style="list-style-type: none"> <li>• 0.09090909090909091 ==&gt; C3 quarter</li> <li>• 0.14545454545454545 ==&gt; G6 quarter</li> <li>• 0.01818181818181818 ==&gt; C4 quarter</li> <li>• 0.05454545454545454 ==&gt; D4 quarter</li> <li>• 0.01818181818181818 ==&gt; E4 quarter</li> <li>• 0.01818181818181818 ==&gt; C7 quarter</li> <li>• 0.03636363636363636 ==&gt; A4 quarter</li> </ul>
24	<ul style="list-style-type: none"> <li>• F4 quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 0.07692307692307693 ==&gt; B5 quarter</li> <li>• 0.11538461538461539 ==&gt; G6 quarter</li> <li>• 0.17307692307692307 ==&gt; G4 quarter</li> <li>• 0.057692307692307696 ==&gt; D5 quarter</li> <li>• 0.28846153846153844 ==&gt; E5 quarter</li> <li>• 0.07692307692307693 ==&gt; A3 quarter</li> <li>• 0.038461538461538464 ==&gt; F4 quarter</li> <li>• 0.019230769230769232 ==&gt; F5 quarter</li> <li>• 0.07692307692307693 ==&gt; C3 quarter</li> <li>• 0.019230769230769232 ==&gt; C7 quarter</li> <li>• 0.019230769230769232 ==&gt; B4 quarter</li> <li>• 0.019230769230769232 ==&gt; G3 quarter</li> <li>• 0.019230769230769232 ==&gt; D4 quarter</li> </ul>
25	<ul style="list-style-type: none"> <li>• G6 quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 0.2 ==&gt; F4 quarter</li> <li>• 0.1 ==&gt; D5 quarter</li> <li>• 0.075 ==&gt; B4 quarter</li> <li>• 0.2 ==&gt; A3 quarter</li> <li>• 0.1 ==&gt; G6 quarter</li> <li>• 0.025 ==&gt; C4 quarter</li> <li>• 0.05 ==&gt; B5 quarter</li> <li>• 0.025 ==&gt; C7 quarter</li> <li>• 0.025 ==&gt; F5 quarter</li> <li>• 0.05 ==&gt; E5 quarter</li> <li>• 0.025 ==&gt; D4 quarter</li> <li>• 0.025 ==&gt; A5 quarter</li> <li>• 0.025 ==&gt; G4 quarter</li> <li>• 0.05 ==&gt; C3 quarter</li> <li>• 0.025 ==&gt; G1 quarter</li> </ul>
26	<ul style="list-style-type: none"> <li>• A3 quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 0.18421052631578946 ==&gt; B5 quarter</li> <li>• 0.02631578947368421 ==&gt; F5 quarter</li> <li>• 0.10526315789473684 ==&gt; D5 quarter</li> <li>• 0.15789473684210525 ==&gt; F4 quarter</li> <li>• 0.15789473684210525 ==&gt; G6 quarter</li> <li>• 0.07894736842105263 ==&gt; C3 quarter</li> <li>• 0.15789473684210525 ==&gt; E5 quarter</li> <li>• 0.02631578947368421 ==&gt; E4 quarter</li> <li>• 0.05263157894736842 ==&gt; A3 quarter</li> <li>• 0.02631578947368421 ==&gt; G4 quarter</li> <li>• 0.02631578947368421 ==&gt; C7 quarter</li> </ul>
27	<ul style="list-style-type: none"> <li>• D5 quarter</li> </ul>	<ul style="list-style-type: none"> <li>• 0.021739130434782608 ==&gt; A4 quarter</li> <li>• 0.08695652173913043 ==&gt; C3 quarter</li> <li>• 0.15217391304347827 ==&gt; D5 quarter</li> <li>• 0.13043478260869565 ==&gt; F4 quarter</li> <li>• 0.08695652173913043 ==&gt; B5 quarter</li> <li>• 0.17391304347826086 ==&gt; E5 quarter</li> <li>• 0.021739130434782608 ==&gt; E4 quarter</li> <li>• 0.06521739130434782 ==&gt; G6 quarter</li> <li>• 0.043478260869565216 ==&gt; B4 quarter</li> <li>• 0.043478260869565216 ==&gt; D4 quarter</li> </ul>

---

	<ul style="list-style-type: none"> <li>• 0.043478260869565216 ==&gt; B3 quarter</li> <li>• 0.043478260869565216 ==&gt; C2 quarter</li> <li>• 0.021739130434782608 ==&gt; A3 quarter</li> <li>• 0.021739130434782608 ==&gt; G4 quarter</li> <li>• 0.021739130434782608 ==&gt; A5 quarter</li> <li>• 0.021739130434782608 ==&gt; REST half Dots: 1</li> </ul>
<b>28</b>	<ul style="list-style-type: none"> <li>• REST half Dots: 1</li> <li>• 1 ==&gt; FIN</li> </ul>

---

This example presents an interesting combination of the two main techniques that constitute the core of this thesis. One can try to create several different grammars with different probabilities and then convert them into Markov chains to create interesting music material that can be later used in a composition.



### **3.5. Summary**

This chapter has described the Stochastic Composer program, the process of creating this application, as well as a tutorial about its use, and an analysis of the sample results. The outcomes presented here constitute only a small amount of the material that was generated. At the end of the testing process 130 samples had been produced by five composers from different background and musical tastes.

It must be made clear that this application was not intended to produce a finished composition. Its main purpose was to generate material with a controllable degree of consistency that could be later used by a composer. The only constraint is the composer's creativity and ingenuity in creating different chains and grammars.

There are several possible improvements to the program. The main one is to incorporate rhythmic elements on a larger scale. One might want to generate chains, for example, only in the rhythmic sphere. Another possible improvement would be to add triplet divisions in the Markov-chain section. Currently the programme only accepts whole notes, half notes, quarters, eighth, sixteenth and thirty-second notes and the dotted variations. In the future it is hoped to add the possibility of working with more note lengths, to make more complex rhythms possible.

Another possible development, though very challenging, is to provide a preview of outcomes from web grammars in staff notation. This requires resolving several questions, like which layer is supposed to be the higher in the hierarchy, or how the rhythm should be created. This will be attempted in the next update of the program.

Stochastic Composer received an especially warm welcome from all the participants. Some of them declared their intention to use the program after the experiment. This will be possible, and the author of this thesis will provide support whenever needed.

## Final remarks

The purpose of this study was to analyse how stochastic processes and probability theory can be used in music. As part of the thesis the author created software that provides tools for composing and analysing stochastic music.

The study was approached by separating the thesis into two parts. The first focuses purely on theoretical aspects of the topic, while the second describes practical experiments carried out by using bespoke software – Stochastic Composer, which was also created as part of the thesis.

In part one, the first theoretical chapter concentrates on basic principles of Markov Chains and analyses the uses of chains in music. The second theoretical chapter is devoted to generative grammars. This chapter first outlines the terminology and provides insight into the mathematical and linguistic origins of the grammars. Providing such in-depth analysis of both processes was crucial to understanding them before turning to the design of the Stochastic Composer.

Chapter three documents the process of creating the software and describes the algorithms that reside in the core of the program. The application is an integral part of the thesis and was created based on the information obtained from first two chapters. Five young composers were asked to use and test the programme, and their results were also presented and analysed in this third chapter.

Music generated by means of the stochastic processes that were implemented in the web application is a proof-of-concept and demonstrates that the topics underlying the whole thesis are not merely theoretical considerations but can be of practical use. The application balances automatic processes with personal taste; although the examples generated are created by the software, it is the user who determines how the application will work. The crucial part in creating musical material is to set the rules that the application will apply; thus it is the user-composer who has the decisive hand in determining the output of the process.

The Stochastic Composer, therefore, constitutes a useful compositional tool that produces output based on stochastic processes and probability theory without requiring a composer to have extensive knowledge of these particular fields of mathematics. As always, however, the application can be further developed and improved. The results of the tests have been very promising and prove that this kind of software can be an important tool in a composer's workshop.

## Bibliography

- Allan, M., & Williams, C. K. (2004). Harmonising Chorales by Probabilistic Inference. *Neural Information Processing Systems Conference*.
- Almagor, H. (1983). A Markov analysis of DNA sequences. *Journal of theoretical biology*, 104(4), 633-645.
- Ames, C. (1989). The Markov Process as a Compositional Model: A Survey and Tutorial. *Leonardo*, 22(2), 175-187.
- Basharin, G., Langville, A., & Naumov, V. (2004, July). The life and work of AA Markov. *Linear Algebra and its Applications*, 386, 3-26.
- Beran, J. (2004). *Beran, Jan. Statistics in musicology* (Vol. 12). CRC Press.
- Brooks, F., Hopkins, A., Neumann, P., & Wright, W. (1957, September). An Experiment in Musical Composition. *Electronic Computers, IRE Transactions on*, 175-182.
- Brzezniak, Z., & Zastawniak, T. (1999). *Basic stochastic processes: a course through exercises*. Springer.
- Chomsky, N. (1956). Three models for the description of language. *IT-2*, pp. 113-124. IRE Transactions on Information Theory.
- Chomsky, N. (1969). *Aspects of the Theory of Syntax*. MIT Press.
- Chomsky, N. (1979). *Immanuel Kant Lectures in Philosophy*. Stanford University.
- Chomsky, N. (2002). *Syntactic structures*. Walter de Gruyter.
- Díaz-Jerez, G. (2015, 11). *FraztMus*. Retrieved from <http://www.gustavodiazjerez.com/>
- Essl, K. (2015, 11). *Karlheinz Essl. Composer / Performer*. Retrieved from <http://www.essl.at/software.html>
- Farbood, M., & Schoner, B. (2001). Analysis and synthesis of Palestrina-style counterpoint using Markov chains. *Proceedings of the International Computer Music Conference*.
- Fernández, J., & Vico, F. (2014). AI methods in algorithmic composition: a comprehensive survey.
- Gardner, M. (1970). Mathematical games: The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, 120-123.
- GDG. (2015, 11). *Conceptual Algorithmic Music*. Retrieved from <http://conceptual-algorithmic-music.blogspot.co.uk>.

- Halliday, M. (2003). Introduction: On the "architecture" of human language. In J. Webster (Ed.), *On Language and Linguistics*. (Vol. 3). London and New York: Continuum.
- Hayes, B. (2013). First links in the Markov chain. *American Scientist*, 101(2), 92-97.
- Hiller, L., & Isaacson, L. (1956). *Illiad suite, for string quartet*. University of Illinois at Urbana-Champaign .
- Hiller, L., & Isaacson, L. M. (1979.). *Experimental Music; Composition with an electronic computer*. Greenwood Publishing Group Inc.
- Holtzman, S. R. (1981). Using generative grammars for music composition. *Computer Music Journal*, 5(1), 51-64.
- Jeppesen, K. (1931). *Counterpoint: the polyphonic vocal style of the sixteenth century*. New York: Dover Publications.
- Jones, K. (1980). *Computer assisted application of stochastic structuring techniques in musical composition and control of digital sound synthesis systems*. London: City University.
- Jones, K. (1981, Summer). Compositional Applications of Stochastic Processes. *Computer Music Journal*, 5(2), 45-61.
- Jones, P., & Smith, P. (2001). *Stochastic processes*. Arnold.
- Kaliakatsos-Papakostas, M. A., Epitropakis, M. G., & Vrahatis, M. N. (2001). Weighted Markov Chain model for musical composer identification. *Applications of Evolutionary Computation*. (pp. 334-343). Springer Berlin Heidelberg.
- Katz, J., & Pesetsky, D. (2011). *The identity thesis for language and music*. Retrieved 06 20, 2014, from URL <http://ling.auf.net/lingBuzz/000959>
- Kemeny, J. G., Snell, J. L., & Knapp, A. (2011). *Markov Chains*.
- Kolberg, O. (1961). *Dziela wszystkie, tom I: Piesni ludu polskiego*. Kraków: PWM, Ludowa Spółdzielnia Wydawnicza.
- Lerdahl, F., & Jackendoff, R. (1985). *A generative theory of tonal music*. MIT press.
- Liu, Y.-W., & Selfridge-Field, E. (2002). *Modeling Music as Markov Chains: Composer Identification*.
- Markov, A. A. (1913). Primer statisticheskogo issledovaniya nad tekstem "Evgeniya Onegina". *Izvestiya Akademii Nauk*.

- McAlpine, K., Hoggar, S., & Miranda, E. (1999). Making Music with Algorithms: A Case-Study System. *Computer Music Journal*, 23(2), 19-30.
- McCormack, J. (1996). Grammar Based Music Composition. In R. J. Stocker, *Complex Systems 96: From Local Interactions to Global Phenomena* (pp. 320-336). ISO Press.
- Meyerhoff, M. (2011). *Introducing sociolinguistics*. Taylor & Francis.
- Miranda, E. R. (2001). *Composing music with computers* (Vol. 1). Taylor & Francis.
- Newman, E. B. (1951). The pattern of vowels and consonants in various languages. *The American journal of psychology*, 369-379.
- Nierhaus, G. (2009). *Algorithmic Composition. Paradigms of Automated Music Generation*. Vien: SpringerWienNewYork.
- Oenbring, R. (2009). *Scientific Rhetoric and Disciplinary Identity: A Critical Rhetorical History of Generative Grammar*. ProQuest.
- Pankin, M. D. (n.d.). *Baseball as a Markov Chain*. Retrieved March 10, 2014, from <http://www.pankin.com/markov/intro.htm>.
- Pfaltz, J., & Rosenfeld, A. (1969). Web grammars. *Proceedings of the 1st international joint conference on Artificial intelligence*. Morgan Kaufmann Publishers Inc.
- Pollastri, E., & Simoncelli, G. (2001). Classification of melodies by composer with hidden markov models. *Web Delivering of Music, 2001. Proceedings. First International Conference*.
- Puterman, M. L. (2009). *Markov decision processes: discrete stochastic dynamic programming* (Vol. 414). John Wiley & Sons.
- Roads, C., & Wieneke, P. (1979). Grammars as representations for music. *Computer Music Journal*, 48-55.
- Romanovsky, V. I. (1970). *Discrete Markov Chains*. (E. Seneta, Trans.) Groningen: Wolters-Noordhoff.
- Schbath, S., Prum, B., & de Turckheim, E. (1995). Exceptional motifs in different Markov chain models for a statistical analysis of DNA sequences. *Journal of Computational Biology*, 2(3), 417-437.
- Shannon, C. E. (1948). A Mathematical Theory of Communication. *The Bell System Technical Journal*, 27, 379-423.

- Steedman, M. (1996). The blues and the abstract truth: Music and mental models. In *Mental models in cognitive science* (pp. 305-318). Psychology Press.
- Taube, R. (2015, 11). *Common Music*. Retrieved from <http://commonmusic.sourceforge.net/>
- Verbeurgt, K., Dinolfo, M., & Fayer, M. (2004b). Extracting Patterns in Music for Composition via Markov Chains. In *Innovations in Applied Artificial Intelligence. 17th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE 2004, Ottawa, Canada, May 17-20, 2004. Proceedings* (pp. 1123-1132). Ottawa: Springer Berlin Heidelberg.
- Verbeurgt, K., Fayer, M., & Dinolfo, M. (2004a). A Hybrid Neural-Markov Approach for Learning to Compose Music by Example. *Virtual Math Museum*. (n.d.). Retrieved 04 2, 2014, from <http://virtualmathmuseum.org/Surface/torus/torus.html>
- Wołkiewicz, J., Kulka, Z., & Kešelj, V. (2007). *N-gram-based approach to composer recognition*. Warsaw: Warsaw University of Technology.
- Yi, L., & Goldsmith, J. (2007). Automatic Generation of Four-part Harmony. *Proceedings of the Conference on Uncertainty in Artificial Intelligence*.