Heretic: A New Live Algorithm

A Thesis

Submitted to the Faculty

in partial fulfillment of the requirements for the

degree of

Masters of Arts

in

Digital Musics

by Hunter Brown

Guarini School of Graduate and Advanced Studies

Dartmouth College

Hanover, New Hampshire

May/2019

Examining Committee:

(chair) ***Michael Casey, Ph.D.***

Taylor Ho Bynum, M.A.

Oliver Bown, Ph.D.

F. Jon Kull, Ph.D.
Dean of the Guarini School of Graduate and Advanced Studies

Abstract

Heretic is an artificially intelligent computer music system to be used within the context of human-machine free improvisation. *Heretic* is written in the SuperCollider programming language with specific aspects of the system implemented in the machine learning software Wekinator. The motivation behind *Heretic's* inception was to create an autonomous system that uses my own improvisational methodology as a computational and conceptual framework for machine improvisation. To achieve this, *Heretic's* architecture is divided into three interdependent modules: *Interpretive Listening, Contextual Decision Making,* and *Musical Synthesis*. Each of these modules serves an important musical function for the spontaneous creation of novel improvised music. When working collectively, these modules are a complete computational model of my improvisational methodology that interacts with human improvisers in real-time. *Heretic* is trained on my approach to improvisation, but through its interactions with a human performer, *Heretic's* own improvisational voice and modes of musical expression emerge. This thesis provides a background and history of previous autonomous computer improvisers and how these systems were developed, thus creating a context for my approach to *Heretic's* development. This thesis also describes improvisational and compositional methodologies that form a basis for my own improvisational methodology and *Heretic's* design. Following this overview of methodologies, I discuss specific musics that formed my approach for crafting *Heretic's* sonic aesthetic and musical voice. Subsequent sections discuss how these concepts and aesthetics are implemented within *Heretic* using machine listening, machine learning, and interactive computer music techniques. Finally, I detail how *Heretic* was tested with aesthetic consideration, provide a formal analysis of music created with *Heretic*, provide conclusions yielded from my performances with *Heretic*, and discuss future work that will develop from this research.

Acknowledgments

Firstly, I would like to thank my thesis readers Michael Casey, Taylor Ho Bynum, and Oliver Bown for their time, support, and expertise throughout the process of creating this thesis. I am also grateful for the dedicated teachers who has profoundly influenced my work as a composer, percussionist, and improviser throughout my musical education: Michael Rosen, Joo Won Park, Aurie Hsu, Peter Swendsen, Tom Lopez, Michelle Lou, Grace Leslie, Amy Garapic, Spencer Topel, Sunny Nam, Seth Parker Woods, and Tomas Fujiwara. A special thanks to my friends who have been there to share their incredible perspectives on life and music: Christy Rose, Lloyd May, Aaron Karp, Clara Allison, Dominic Coles, Josh Davis, Darius Sadighi, and many others. Briana Krewson, thanks for teaching me how to write, you deserve a medal. Yvonne Pelletier, thank you for your help, making me laugh, and sharing your wisdom. Emma Blackman, this thesis certainly wouldn't have been possible without your care and endless support, thanks for always making me smile. Last but not least, thanks and love to my incredibly supportive family, Mom, Dad, Nan, Granddad, and Blaize, I'm proud to be a "chip off the old block."

Contents

List of Figures

1 Introduction

Heretic is an artificially intelligent computer music system to be used within the context of human-machine free improvisation. *Heretic* is written in the SuperCollider programming language with specific aspects of the system implemented in the machine learning software Wekinator [26, 78]. The motivation behind *Heretic's* inception was to create an autonomous system that uses my own improvisational methodology as a computational and conceptual framework for machine improvisation. To achieve this, *Heretic's* architecture is divided into three interdependent modules: *Interpretive Listening, Contextual Decision Making,* and *Musical Synthesis*. Each of these modules serves an important musical function for the performance of novel improvised music. When working collectively, these modules are a complete computational model of my improvisational methodology that interacts with a human improviser in real-time. *Heretic* is trained on my approach to improvisation, but through its interactions with a human performer, *Heretic's* own improvisational voice and modes of musical expression emerge.

Like any human improviser, *Heretic* behaves autonomously. In term's of Robert Rowe's definitions of "player" and "instrument" paradigms within interactive computer music systems, *Heretic's* functionality fits the category of a "player" program [63]. *Heretic* is not an instrument to be controlled by a performer; *Heretic* generates and controls its own musical output by listening to the present musical moment, interpreting what it hears, making a musical decision based on this information, and then synthesizing its decision as music. This approach to teaching *Heretic* how to improvise is analogous to many methodologies within improvisation pedagogy [5]. The teacher provides a student with a musical tool-kit that is based on the teacher's own knowledge and experience. The student then uses this tool-kit as a basis for addressing their own needs within any improvisational context. I have provided *Heretic* with a methodology (or a tool-kit), and *Heretic* autonomously implements this methodology according to its own perception of the current musical context. This is how an improviser develops their own voice or sound, by learning what has come before them and interpreting this information through their own experiential lens. As eloquently stated by guitarist Joe Morris, "you can't play what you don't know" [44].

Systems similar to *Heretic* often prioritize agnostic approaches to machine improvisation by avoiding prior musical knowledge in the system's training stage. These system's creators assert that a system's autonomy and novelty are comprised with knowledge-based or grammatical approaches to improvisational decision making [34, 51, 83]. However, prominent improvisers such as Cecil Taylor, Ornette Coleman, Joe Morris, and Anthony Braxton detail their approaches to improvisation as *languages* or grammatical systems [2, 9, 41, 73]. These improvisers contextualize the real-time musical materials of their band-mates by applying their formulated grammatical systems to their decision making processes. I argue that a grammatical approach for organizing an improvisational vocabulary yields idiosyncratic interactions, complete machine autonomy, and novel musical output when used within the context of human-machine improvisation.

While *Heretic* is trained on my methodology as a human improviser, I intend for *Heretic* to maintain its intrinsic computer-like traits as a means of achieving "mutual subversion" [5]. Mutual subversion is when a common musical language between two or more improvisers is so collectively well-known, that the musicians are able to unpredictably test the confines of this language by challenging the musical decisions of other players. Derek Bailey defines "mutual subversion" in his seminal text *Improvisation: its nature and practice in music*, "a common language [will] come into being, and a mutual trust in each other permits one to push against the limitations of that language..." [5]. While *Heretic* can hear my language and interpret its contextual meaning in real-time, *Heretic* also implements stochastic processes when making decisions. In his book *Machine Musicianship*, Robert Rowe identifies a successful machine improviser as a system that is able to "contribute a convincing musical voice in a completely unstructured and unpredictable environment" [62]. Therefore, *Heretic's* carefully implemented Markov models enable unpredictable stochasticism that achieves Bailey's concept of mutual subversion, while also "contributing a convincing musical voice" to an improvised context [5, 62]. Using stochasticism to challenge a human's skill as an improviser and the musical limits of an improvised performance are my main motivations for designing an artificially intelligent computer music system. A human's spontaneous musical instincts combined with a machine's autonomous, computer-like personality results in novel and idiosyncratic music that bridges the sonic worlds of the human and machine.

In this thesis, I first provide a background and history of previous autonomous computer improvisers and how these systems were developed, thus creating a context for my approach to *Heretic's* development. Also in this thesis, I describe improvisational and compositional methodologies that form a basis for my own improvisational methodology and *Heretic's* design. Following this overview of methodologies, I discuss specific musics that formed my approach for crafting *Heretic's* sonic aesthetic and musical voice. Subsequent sections discuss how these concepts and aesthetics are implemented within *Heretic* using machine listening, machine learning, and interactive computer music techniques. Finally, I detail how *Heretic* was tested with aesthetic consideration, provide a formal analysis of the music created with *Heretic*, provide conclusions yielded from my performances with *Heretic*, and discuss future work that will develop from this research.

2 Background

Heretic's conceptual foundation centers around three interconnected sectors of research: Live Algorithms, improvisational methodologies, and electroacoustic Music. Live Algorithms research within the computer music field focuses on autonomous music systems capable of human-compatible performance [6]. Through studying the practice and computational methods behind developing Live Algorithms, specifically systems created by Tim Blackwell, Oliver Bown, Michael Young, George Lewis, and IRCAM's Music Representations Team, I found that creating a Live Algorithm of my own required researching improvisational methodologies that fit my musical and aesthetic goals for *Heretic's* sonic voice and interactive properties [8, 35, 37, 83].

I have applied these methodologies to my own improvisational practice and to creating *Heretic*. Specifically, *Heretic's* functionality incorporates theoretical concepts from Joe Morris' *The Properties of Free Music*, Derek Bailey's *Improvisation: Its Nature and Practice in Music*, and Anthony Braxton's *Language Music* [5,9,39,44]. *Heretic* also uses concepts proposed by Denis Smalley and Curtis Roads in their respective texts *Spectromorphology: Explaining Sound-Shapes*, and *Microsound* [59,70]. Smalley and Roads' research in the domain of electroacoustic music serves as a conceptual link between Braxton and Morris' improvisational methodologies and *Heretic's* emergent sonic properties via electroacoustic sound worlds. Finally, my study of free improvisation facilitated the discovery of own sonic aesthetic as an improviser and composer. I have particularly found sonic inspiration in recent improvised electro-acoustic music, feedback electronics, free jazz, and Japanese Onkyokei music. Later in this chapter, I discuss three albums that are representative of the aesthetic inspiration found in these musical traditions. These albums include *Event Horizon* - Peter Evans and Sam Pluta, *Centering and Displacement* - Frank Rosaly, and *Color Quanta* - Toshimaru Nakamura and Kim Cascone.

Heretic's conceptual foundation is a direct result of my experience as a percussionist, improviser, and composer. I have extracted and synthesised various aspects from these improvisational and compositional methodologies throughout my creative development as a musician. *Heretic* is a combined manifestation of these methodologies resulting in an abstract expression of my musical voice. I see this abstract expression of my musical voice not as a direct mirror into my own improvisational practice, but rather a method of enabling a *common language* between myself and *Heretic* [5]. *Heretic* demonstrates how a shared improvisational language can be used as collaborative infrastructure between the human performer and machine to spontaneously create ever-changing musical frameworks and gestures.

2.1 Live Algorithms

This section defines and explores Live Algorithms. First, Tim Blackwell, Oliver Bown, and Michael Young's research in autonomous computer improvisers provides a definition of Live Algorithms, and various methods for creating Live Algorithms [7]. Second, I examine and compare two Live Algorithms, George Lewis' *Voyager* and IRCAM's Music Representations Team's (RepMus) *OMax*. In this examination, I use Blackwell et al's definition of a Live Algorithm to describe the conceptual frameworks behind these systems' designs, the precise intentions behind their creation, how these intentions are implemented computationally, and the music that emerges from these systems' respective implementations.

2.1.1 Live Algorithms Research

In their paper *Live Algorithms: Towards Autonomous Computer Improvisers*, Tim Blackwell, Oliver Bown, and Michael Young define a Live Algorithm as "an autonomous music system capable of human-compatible performance. The context is improvised music; the Live Algorithm listens, reflects, selects, imagines and articulates its musical thoughts as sound in a continuous process" [6]. Their original motivation behind this research was "[to] emulate human performance convincingly" [6]. However, the scope of Live Algorithms research now extends beyond simply emulating human performance, Blackwell et al see Live Algorithms "as a contributing and creative group member with the same musical status as any other performer... The overarching aim is to extend the possibilities of music performance." [6].

Blackwell et al. also assert that a system must include qualities of "autonomy, novelty, participation and leadership" to be considered a Live Algorithm [6]. Blackwell et al define *autonomy* as the ability for the machine "to act and respond to unknowable and unforeseen inputs in ways that have not be completely prescribed" [6]. *Novelty* is achieved by "[avoiding] the cliched and the obvious" when "supporting, leading, or subverting" other musicians [6]. The quality of *participation* consists of "supporting ongoing musical activity by making contributions that do not detract from but rather enhance the current musical direction" [6]. Live Algorithms obtain *leadership* by "[attempting] to change the musical direction, to invoke a new musical center" [6].

In order to design a system that adheres to these four qualities, Blackwell et al describe a design methodology for creating a Live Algorithm, the *PQF* architecture. The *PQF* architecture contains three modules that are used in various combinations to develop computer music systems. These modules describe three basic computational functionalities, and are the literal software components of a Live Algorithm. The modules are defined as follows: *P* (listening/analysis), *Q* (performing/synthesis), *F* (patterning, reasoning or even intuiting) [6, 7].

P functions as a real-time music information retrieval module, *Q* is a sound making module, and *F* is the decision making module. *P* and *Q* are common within computer music, however *F*, the autonomous decision making module, is what furthest separates Live Algorithms from other sectors of computer music. *Heretic's* architecture is modelled after Blackwell et al's proposed architecture. *Heretic's P* module is defined as *Interpretive Listening*, the *Q* module as *Musical Synthesis*, and the *F* module as *Contextual Decision Making*.

To Blackwell et al, a Live Algorithm is not complete unless "these three modules are present, interconnected, absent of a human controller, and such that the above four characteristics (autonomy, novelty, participation and leadership) are ascribable attributes of the system" [6]. Through Blackwell et al's research in the domain of human-machine free improvisation, I am able to design *Heretic* using their proven formalized methodology of Live Algorithm development as a conceptual and computational benchmark. By using their proposed architectural design scheme, and by evaluating *Heretic* according to their proposed qualities that define a Live Algorithm, "autonomy, novelty, participation and leadership," I am able to imbue *Heretic* with my own aesthetic values as an improviser while also ensuring *Heretic's* autonomy, creativity, and "musical coherence" [62]. In Chapter 4 of this thesis, I will use Blackwell et al's criteria for a Live Algorithm as guidelines for evaluating a recorded improvisation between myself and *Heretic*.

2.1.2 Previous Live Algorithms: *Voyager*

The first known Live Algorithm is George Lewis' *Voyager*. Lewis developed the first version of *Voyager* between 1986 and 1988 at the Studio for Elektro-Instrumentale Muziek (STEIM) in Amsterdam. *Voyager* was written in dialectics of the Fourth programming language and the current version is written in Max/MSP [77]. Lewis describes *Voyager* as:

"a nonhierarchical, interactive musical environment that privileges improvisation... improvisers engage in dialogue with a computer-driven, interactive 'virtual improvising orchestra.' A computer program analyzes aspects of a human improviser's performance in real time, using that analysis to guide an automatic composition..." [37].

Lewis' goal in creating *Voyager* was to imbue the system with a specific component of his musical aesthetic. As stated by Lewis, "Musical computer programs, like any texts, are not 'objective' or 'universal,' but instead represent the particular ideas of their creators" [37]. The "particular idea" Lewis focused on in *Voyager's* development is derived from an aesthetic associated with African visual art. "Multidominance" is a term coined by the visual artist Robert L. Douglas to describe "the multiple use of colors in intense degrees, or the multiple use of textures, design patterns, or shapes," or in musical terms, "the predisposition to use multiple types of rhythm in musical construction speaks equally to a distinct aesthetic as does the multiple use of visual elements" [37]. Lewis' choice for *Voyager* to embody a sonic aesthetic of multidominance is a reaction against Eurocentric notions of composition and improvisation prominent in the creation of computer music systems, a reflection of Lewis' comparative research in "Afrological" and "Eurological" improvisational practices [36]. In Lewis' own words "sudden changes of mood, tempo and orchestration, [eschew] the slowly moving timbral narratives characteristic of much institutionally based computer music... *Voyager* is in clear violation of the dictum that Douglas identifies here as Eurocentric: 'Don't overcrowd your composition with too many elements.'" [37].

In terms of Blackwell et al's *PQF* architecture, Lewis designs each of *Voyager's* seperate modules to yield an aesthetic of multidominance. *Voyager* interacts with the human's real-time pitch data via "a set of 64 asynchronously operating single-voice MIDI-controlled 'players,' all generating music in real time... moving in and out of metric synchronicity, with no necessary arithmetic correlation between the strongly discursive layers of multirhythm," which is an an essential quality to Douglas' multidominance [37]. Lewis defines the human interactions with *Voyager's* decision making and corresponding musical output as "emotional transduction" [37]. Emotional transduction is a feedback loop between the human's musical response to *Voyager's* output, and *Voyager's* musical response to the human performer. The human and *Voyager* are constantly listening to each other, an inherent phenomenon in all Live Algorithms [38].

The potential for the human performer and *Voyager* to influence musical change at a rate of five to seven seconds allows for "dense, rapid accretions of sonic information" which is inherent to Lewis' aesthetic of multidominance [37]. The sonic breadth of *Voyager's* output, stemming from the large number of musical voices present, those musical voices' number of sonic manipulations, the potential for quick shifts in musical character, and the bi-directional nature of interactivity epitomizes Lewis' intended aesthetic of multidominance. *Voyager's* design and resulting musical output adheres to Lewis' aesthetic intention, and Blackwell et al's four qualities of "autonomy, novelty, participation and leadership," making *Voyager* a uniquely expressive Live Algorithm.

2.1.3 Previous Live Algorithms: *Omax*

The first version of *OMax* was released by IRCAM in 2004 and new iterations of this system are currently in development under the names of *SoMax* and *ImproteK* [34, 50]. The most recent version is written entirely in Max/MSP, and the original version was written in OpenMusic and Max/MSP, hence the name *Omax*. *Omax's* creators describe the system as:

"...an improvising software which uses on-the-fly learning and generating from a live source. It is capable of simultaneously listening to the audio stream of a musician, extracting information from that stream, modeling this information into a complex formal structure (think of a musical roadmap), then navigating this structure by recombining the musical material to create variation(s) or 'clone(s)' of the input" [33].

RepMus' principal intention in creating *Omax* reflects their opinion of what "makes an interesting improvisation," by making "clones" ("or stylistic models") of the human performer [34]. They believe in order for an improvisation to be "interesting," it must adhere to a consistent musical style derived from the immediate musical material provided by the human performer [33].

Omax's implementation relies on machine listening, and a form of interactivity called "stylistic reinjection" [35]. Stylistic reinjection is similar to Lewis' concept of "emotional transduction," in that it is a bi-directional feedback loop consisting of the computer analyzing the human's playing, the computer generating musical output based on this analysis, and the human reacting to the computer's musical output.

Omax's musical output, or *Q* module, consists of samples collected from the human performer that are rearranged in real-time. *Omax's P* and *F* modules interact to generate "clones" by organizing the human's recorded samples via pitch and/or spectral analysis, and then outputting these clones in response to an analysis of the human's real-time playing. *Omax's P* module uses monophonic pitch or timbral tracking to slice the recorded audio into "micro-units" [35]. These micro-units are determined by the "birth and death of significant events," or the beginning and ends of musical phrases [35]. These sliced micro-units are then grouped into larger "macro-units," based on their pitch or timbral similarity. *Omax* then "time stamps" these macro-units, determining when each macro-unit will be "reinjected" into the musical performance. The process of time stamping macro-units comes from an anticipatory algorithm referred to as a "factor oracle" [16]. Using Hidden Markov models, the factor oracle anticipates the potential musical sequencing of the improvisation based on the audio features extracted from the human musician's real-time playing [4].

This model of interactivity generated by the *F* module, involves "recombinations of the original discourse justified by the model and realized by cutting and pasting the original material in real-time" [35]. This is a direct reflection of RepMus' belief that a "balance of recurrence and innovation makes an interesting improvisation" [35]. In this case, the "recurrence" is the use of recorded audio from the human performer, and the "innovation" being the rearrangement of this recorded material via the factor oracle and the use of "stylistic reinjection" [35].

While RepMus states their goal is an "agnostic" system with no stylistic bias, I argue that stylistic bias is impossible when designing a computer music system. Despite this goal of agnosticism, RepMus describes *Omax* using language that contradicts their goal. For example, they state that the "recurrence" of musical materials is what "makes an interesting improvisation" [33]. They also state that Omax can sense the "birth and death of significant events," but how can Omax know whether a musical event is "significant" or not without being programmed to perceive this as their creator would? Whether or not the creator of a Live Algorithm intends it or not, I believe a Live Algorithm's musical output always "represents the particular ideas of their creators," which is why I have intentionally and explicitly programmed *Heretic* to create music using my own approach to improvisation [37].

2.2 Compositional/Improvisational Methodologies

This section details the improvisational and compositional methodologies that have shaped my performance practice as an improviser and *Heretic's Interpretive Listening* and *Con-*

textual Decision Making modules. In regard to *Heretic's Musical Synthesis* module, I will examine musics that have informed my approach to *Heretic's* sound design, and how this sonic aesthetic differs from other Live Algorithms.

2.2.1 Interpretive Listening: Braxton, Smalley, Roads

In the context of collaborative free music, it is important to actively listen to and interpret real-time musical material into an organizational framework. *Heretic's* organizational methodology stems from a combination of Curtis Roads' time scale definitions, Smalley's *Structural Functions* and Anthony Braxton's *Language Music* system [9, 39].

In Curtis Roads' *Microsound*, he provides an overview of musical time scales, and how these temporal definitions function in musical practice. Roads defines the macro-time scale of musical structure as, "corresponding to the notion of form, and [macro-form] encompasses the overall architecture of a composition" [59]. Roads asserts that there are two ways to generate macro-form, top-down or bottom-up. In his words, "A strict top-down approach considers macrostructure as a preconceived global plan or template whose details are filled in by later stages of composition... By contrast, a strict bottom-up approach conceives of form as the result of a process of internal development provoked by interactions on lower levels of musical structure" [59].

The contextualization of low-level musical materials into a higher-level structure within free improvisation is generated via a "bottom-up" approach to organization [59]. Improvisers' ability to organize their band-mates' spontaneous musical materials in real-time dictates how they might musically respond, and how these interactions enable a formal structure to emerge. In other words, improvisers are collectively searching for a balance of self-organization and forward momentum. As Joe Morris states in his text *Perpetual Frontier: The Properties of Free Music*, "Form is either stated or it emerges in process... Modulation of tonal center or meter, dynamics, density, relation and expression of pulse, interactive postures, introduction of different melodic structures, etc., all signify changes in form" [44].

This ability to organize spontaneous musical materials into higher-level structural contexts in real-time is rooted in a complete understanding of the collective and individual musical languages of one's band-mates. As Derek Bailey states, "a common language [will] come into being, and a mutual trust in each other permits one to push against the limitations of that language..." [5]. The practice of free improvisation is often associated with the use of non-traditional musical materials, making this "common language" more difficult to organize than forms of improvisation that operate using more traditional musical devices.

Improvisational idioms such as Bebop, North Indian classical music, and Flamenco, describe low-level materials using traditional organization systems such as melody, harmony, and rhythm. More idiosyncratic forms of improvisation such as free jazz, Onkyokei, or noise music often use nontraditional musical materials that are difficult to abstract into these traditional organizational frameworks [14]. Many practitioners of free improvisation have developed their own organizational frameworks, such as Cecil Taylor's *Unit Structures*, Ornette Coleman's *Harmolodics*, and Joe Morris' *Properties of Free Music* [2, 41, 73]. Anthony Braxton's *Language Music* is particularly effective due to its adaptability to the characteristics of any instrumental and/or improvisational practice [9].

onsets	***continuants***	***terminations***
departure	passage	arrival
emergence	transition	disappearance
anacrusis	prolongation	closure
attack	maintenance	release
upbeat	statement	resolution
downbeat		plane

Figure 2.1: Smalley's *Structural Functions* [70]

With this concept of low-level versus high-level musical organization defined by Roads, Denis Smalley provides description of movement between these high-level structures In his text, *Spectromorphology: Explaining Sound-Shapes* [70]. His concepts of "gesture vs.

texture", and "structural functions" are applied to collaboratively generating formal motion within *Heretic's Contextual Decision Making* module. For example, a musical state that remains static with no momentum towards another state is a texture. A musical state that builds from nothing to utter chaos over the course of a few minutes is an example of a gesture. Within these two broad categories of musical motion, Smalley defines various *structural functions* that can be applied to movement in between these broad descriptors of musical state. For example, an "emergence" could be heard as a state of low-energy material moving towards a state of high energy material. High-energy materials deteriorating to silence is described as a "disappearance" (Figure 2.1). To understand the details of how this macro-formal movement functions within *Heretic*, I will define *Heretic's* low-level organizational structure from which form "emerge[s] in process" [70].

Heretic uses a reinterpretation of Anthony Braxton's *Language Music* system as a means of organizing low-level musical materials into a fluid formal structure as described by Roads and Smalley. Braxton describes his system as a "syntax of musical devices for solo alto saxophone improvisation" [39] (Figure 2.2). Trumpeter Nate Wooley provides a more detailed description:

"In essence, *Language Music* consists of a list of 12 'types' or descriptions of broad musical parameters, which the performer uses to limit their improvisation... It may be more accurate to call them starting points or springboards to musical activity. If taken individually, this system provides a structural framework for improvising, breaking the stream-of-consciousness trope that can weigh down free music" [81].

Using a syntax akin to Braxton's *Language Music* system as applied to listening within an improvisational setting is effective and flexible. Not only can this syntax accurately describe abstract musical materials, but these groups of materials can also be used to define higher-level formal structures within an improvisation. Graham Lock describes Braxton's ability to do exactly that, "Braxton's lecture at the Guildhall is to do with the way in which the pool of 'language types' that he has built up over the years can be employed in the compositional process. He takes one example of his 'language types,' staccato line formings, and demonstrates its various functions for generating form..." [39].

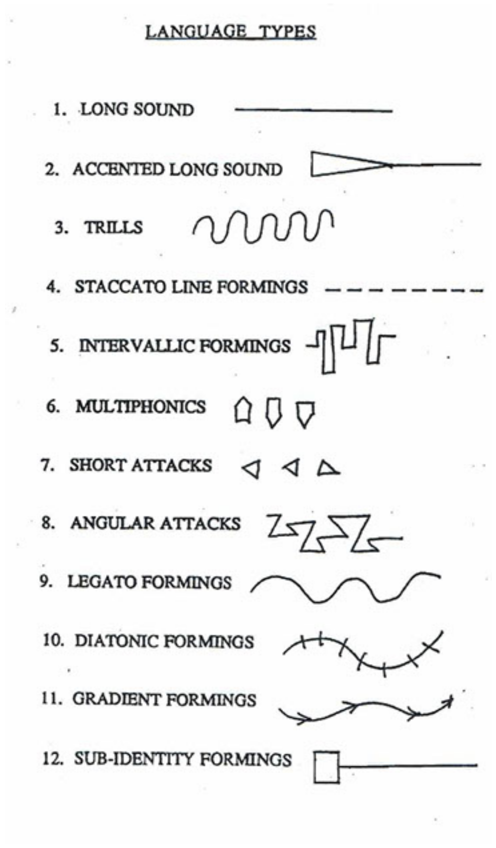


Figure 2.2: Braxton's *Language Types*
[39]

While Braxton's *Language Music* system was designed for pitched instruments, my personal *Language Music* system has been designed for my performance practice as an improviser, percussionist, and electronic musician. My language system is organized into ten language types, and is the basis for training *Heretic's Interpretive Listening* module to achieve "musical coherence" within a formal structure via implementing a "theoretical hypothesis" as suggested by Eigenfeldt et al. in their paper *Flexible Generation of Musical Form: Beyond Mere Generation* (Figure 2.3) [24]. Chapter 3 describes how my *Language Music* system is used to train *Heretic* to hear, recognize, and organize unseen musical material from a human improviser into a formal structure. This *Language Music* system is also used as an organizational framework for *Heretic's Musical Synthesis* module (Section 3.4).

Heretic's version of Braxton's *Language Music* system is akin to syntactical organizational frameworks often used in electroacoustic music composition. According to Curtis Roads' definitions of various musical timescales, Braxton's *Language Music* system would be categorized in the "meso time scale" [59]. In Roads' own words, "The mesostructual level groups sound objects into a quasi hierarchy of phrase structures... the vocabulary of a piece of music" [59].

Trevor Wishart defines the meso time scale as a *sequence* that consists of *fields*, and *orders* [79]. A *field* is a collection of musical materials that adhere to "a particular natural language," and an *order* is how these materials are arranged in time. Denis Smalley's concept of *Spectromorphology* also details a process for organizing low-level musical materials into higher-level typologies [69]. Smalley describes Spectromorphology as, "not a compositional theory or method, but a descriptive tool based on aural perception... It is intended to aid listening... [and] how composers conceive musical content and form their aims, models, systems, techniques, and structural plans" [70]. Like Braxton's *Language Music* system, Smalley's *Spectromorphology* acts as a framework for contextual listening, and as a means of organizing non-traditional musical materials into a structural hierarchy.

Roads, Wishart, and Smalley's theories regarding the high-level organization of non-traditional musical materials resonates with Braxton's *Language Music* system. Understanding the connection between these different organizational systems is essential to connecting the act of improvisational listening to the process of enabling *Heretic* to contextualize the low-level audio feature data from its *Interpretive Listening* module. In particular, connecting Braxton's *Language Music* to the organizational principles of electronic music aids in designing how *Heretic* implements its *own* musical language in dialogue with the human performer's musical language.

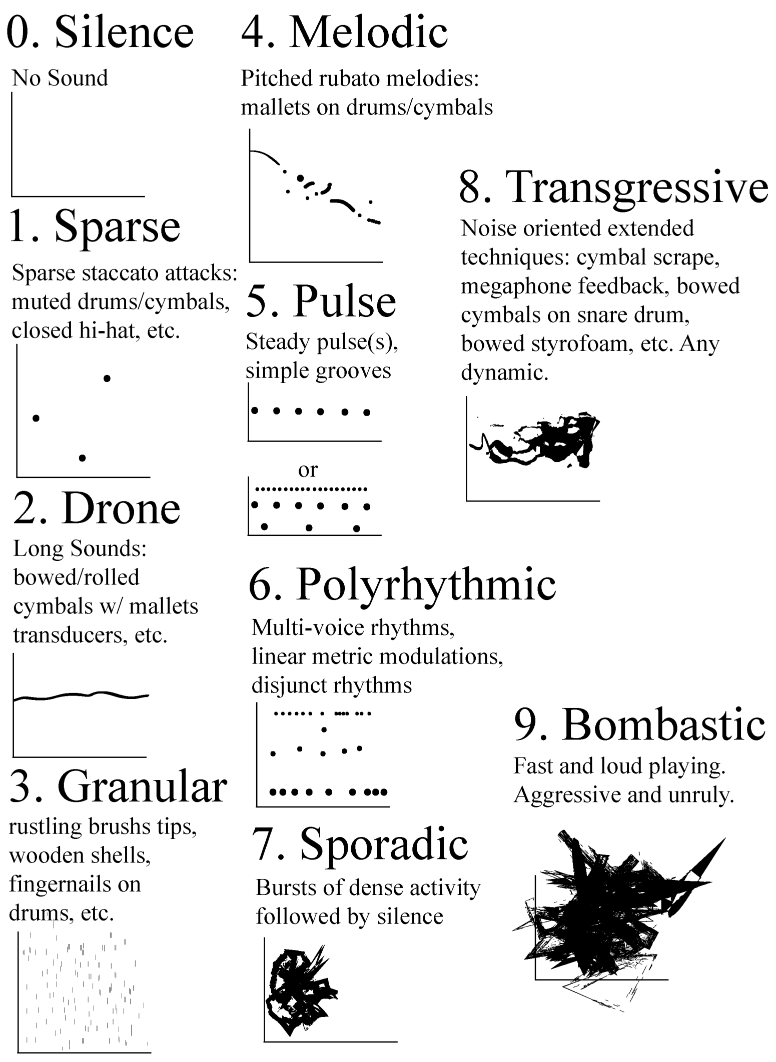

0. Silence
No Sound
4. Melodic
Pitched rubato melodies: mallets on drums/cymbals
8. Transgressive
Noise oriented extended techniques: cymbal scrape, megaphone feedback, bowed cymbals on snare drum, bowed styrofoam, etc. Any dynamic.
1. Sparse
Sparse staccato attacks: muted drums/cymbals, closed hi-hat, etc.
5. Pulse
Steady pulse(s), simple grooves
or
2. Drone
Long Sounds: bowed/rolled cymbals w/ mallets transducers, etc.
6. Polyrhythmic
Multi-voice rhythms, linear metric modulations, disjunct rhythms
9. Bombastic
Fast and loud playing. Aggressive and unruly.
3. Granular
rustling brushs tips, wooden shells, fingernails on drums, etc.
7. Sporadic
Bursts of dense activity followed by silence

Figure 2.3: My reinterpretation of Braxton's *Language Types* [39]

2.2.2 Contextual Decision Making: Morris

The ability to hear and interpret real-time musical materials enables *Heretic* to make musical decisions. *Heretic's Contextual Decision Making Module* is based on a combination of Lewis' *emotional transduction*, Bailey's *mutual subversion* and Joe Morris' *Postures of Interaction*. In regard to *emotional transduction*, *Heretic's* decision making not only affects its own musical output, but it also affects the human's musical output. This is because the human must actively listen to *Heretic* when formulating a musical response. While *Heretic's Contextual Decision Making* module is intended to exude "musical coherence" in its interactions with a human performer, it is also intended to "mutually subvert" the human performer's playing, testing the limits of the common language that emerges between *Heretic* and the human performer [5,62]. This is the logic behind using Markov modelling to design *Heretic's Contextual Decision Making* module. Markov chains are able to create a generalized musical behavior, while maintaining an unpredictable disposition that yields idiosyncratic interactions.

Morris' *postures of interaction* describe both a concise approach for decision making in non-solo free music contexts, and the possible ways one improviser can interact with another during a performance. This collection of interactive postures formulates the basis for *Heretic's Contextual Decision Making* module. These postures are situated in a second-order Markov Chain that generates an internal behavioral state that dictates the low-level aspects of *Heretic's* decision making process (Section 3.3). Morris' *Postures of Interaction* are defined as [44]:

Solo - "A solo is heard as a lead voice. The player plays with the intention that their playing will be heard as a lead voice. Multiple players soloing is also possible."
Unison "Unison can mean playing the same notes, the same rhythm, with the same density, to just at the same time... playing things that are exact or similar can be viewed as unison."
Complement - "Playing in support of a soloist or soloists, complementing the solo statement. This posture depends on the player playing at a dynamic level and with a degree of density that allows the soloist to be head as the lead voice."
Juxtaposition - "Playing something entirely different than another soloist or different than what the ensemble is playing... [Juxtaposition] requires a complex combination of material that differs completely from what is being played by others."
Silence - "Not playing. Not making a sound as an intentional posture." [44]

Morris goes on to describe the musical devices one could implement in order to express these postures in a musical context. One could liken aspects of these musical devices to the *language types* described in section 2.2.1.

Layers: "Playing in dynamic layers that highlight the interactive relationship."
Density: "The proportion of sound made as an intentional decision."
Melody: "Melodic statements used as solo in free music can be meant to display of present melodic structure that can be referred to by other players."
Tempo: "Increasing or decreasing the tempo in an individual or ensemble statement can affect the posture of the interaction among players."
Chords/Clusters: "In complement or juxtaposition, chords and clusters of notes can signify and clarify an interactive posture."
Sustain/Decay/Envelop: "The attack and decay time of a sound and its length of sustain can signify and clarify its posture of interaction among players."
Non-Pitched Sounds/Noise: "Percussive, muted, unconventional uses of an instrument, extended techniques that can make non-pitched sounds and any kind of noise can signify and clarify a posture in the interaction among players."
Rhythmic Displacement and Pulse: "In unison, complement, and juxtaposition, rhythmic displacement, stating or implying pulse and the use of melodic rhythms, can signify, and clarify a posture interaction among players."
Repetition: "Repeating any material to signify and clarify a posture of interaction among players."

Heretic listens to and interprets a human's playing, considers the the current context of what it hears, chooses a *posture of interaction*, and then synthesizes a musical response using its own collection of *language types*. Section 3.3 of this thesis details how a series of embedded Markov chains are used to design *Heretic's Contextual Decision Making* module.

2.3 Musical Synthesis: Pluta/Evans, Rosaly, Nakamura/Cascone

The Live Algorithms described in section 1.2 vary in sonic aesthetic and their approach to audio synthesis. *Voyager* outputs streams of general MIDI sounds, *OMax* uses sampled material from the human performer, and *piano-prosthesis* uses previously sampled piano sounds with real-time electronic processing [35,37,83]. *Heretic's Musical Synthesis* module differs from these systems by expressing its musical language as a comprehensive laptop improviser. *Heretic* applies real-time processing to the sound of the human performer, manipulates audio samples, and controls digital synthesizers. This dynamic and fluid approach to timbre reflects my aesthetic interests in improvised electro-acoustic music, feedback electronics, free jazz, and Onkyokei music. This section contains references to specific albums that I have found to be representative of *Heretic's* sonic ambitions, however these albums are not a comprehensive list of inspirational music. The following albums serve as sonic benchmarks for *Heretic's Musical Synthesis* module via the sonic palette and musical interactions demonstrated by the musicians in these recordings.

2.3.1 Pluta/Evans

Sam Pluta is a composer, sound artist, and laptop improviser based in Chicago Illinois, and Peter Evans is a trumpeter, composer, and improviser based in New York City. They have worked together for many years in various musical contexts, including Evan Parker's Electro-Acoustic Nonet, Peter Evan's Rocket Science, and as a duo. In 2014, Pluta and Evans released an album of live free improvisations entitled *Event Horizons* [56]. The album was released on Pluta's record label, Carrier Records. The album's liner notes read:

"With Event Horizon, Peter Evans and Sam Pluta have created a new musical language. Bridging Evans' efforts to explore uncharted sonic worlds for a traditional acoustic instrument with Pluta's customized and highly flexible software, their instruments are at some points fused into a unified sonic entity indistinguishable as two voices, while at others engaged in violent counterpoint" [56].

Sam Pluta and Peter Evans' duo project is an exemplary case of two musicians cultivating a unified sonic voice. Pluta's ability to sample, transform, and re-contextualize Evans' playing in real-time, along with Evan's virtuosic ability to acoustically reproduce electronic sounds on the trumpet, enables this sonic melding to take a clear shape by means of an interactive feedback loop between the two musicians. "Pluta's customized and highly flexible software," is rich with timbral variety [56]. This software is the subject of Pluta's Doctoral Dissertation at Columbia University, *The Live Modular Software Instrument*, which like *Heretic* is written in the SuperCollider programming language [55]. On this record, Pluta creates a sound palette ranging from bursts of asynchronous micro-sonic glitch, to metrically pulsing noise, all the way to delicately sparse loops. Pluta and Evans' ability to truly achieve a cohesive musical language is extraordinary. At points in this album, it is hard to parse which player is making a particular sound. *Heretic's Musical Synthesis* module uses Pluta's *Live Modular Software Instrument* as a benchmark for range of sonic flexibility and modular design. Pluta and Evans' cohesive sonic aesthetic also serves as inspiration for imbuing *Heretic* with the ability to transform a human performer's sound in a subtle, refined, and vivid manner.

2.3.2 Rosaly

Frank Rosaly is a drummer, composer, electronic musician, and improviser based in Amsterdam, Netherlands. His work is self described as "navigating a fine line between the vibrant improvised music, experimental, rock and jazz communities" [61]. In 2012, Rosaly released in album entitled *Centering and Displacement* on Utech records as a part of his solo project *Milkwork*, which is described as, "a study in the integration of electronically manipulated percussion instruments, improvising dense sound walls with controlled feedback, over-driven amplified drums. Frank blends soundscapes designed through analog electronic synthesis with extended techniques on unamplified drum set" [61].

Rosaly's process in creating this record involved many hours of recording improvised drum material, editing this material, re-sequencing it electronically, and then recording improvised electronics over this recorded drum material. On this album, Rosaly is improvising with himself. This is similar to my approach to training *Heretic* to improvise using my own improvisational methodology as a musical framework. *Centering and Displacement's* sonic aesthetic is a rich combination of unruly noise oriented feedback, and dynamically subtle melodic and rhythmic material. At points the feedback material is clearly disparate from Rosaly's drumming, and at times the drums seem to melt into the surging waywardness of the feedback. It is also clear that Rosaly is attempting to mimic or mirror the quality of the drum material while the electronics maintain a separate voice. *Centering and Displacement* is sonically divergent from Pluta and Evans' record in the clear separation of the drums and electronics in sonic space and musical material. However, there are still moments of clear aesthetic cohesion, in which the electronics always have a mind of their own, a distinct voice that is never directly controlled or governed by the drum material [60]. This reflects my aim of imbuing *Heretic* with its own sonic voice, while also enabling the software to achieve aesthetic cohesion with the human performer.

2.3.3 Nakamura/Cascone

Toshimaru Nakamura is a founder of the Onkyokei music movement that began in Japan during the late 1990s. A scene still active today, Onkyokei music is rooted in free improvisation using electronics, "quiet noise," and "exploring the fine-grained textural details of acoustic and electronic sound" [18]. Nakamura is particularly known for his use of the no-input mixing board as his primary instrument. He details his use of this instrument:

"Toshimaru Nakamura's instrument is the no-input mixing board, which describes a way of using a standard mixing board as an electronic music instrument, producing sound without any external audio input. The use of the mixing board in this manner is not only innovative in the sounds it can create but, more importantly, in the approach this method of working with the mixer demands. The unpredictability of the instrument requires an attitude of obedience and resignation to the system and the sounds it produces, bringing a high level of indeterminacy and surprise to the music" [47].

Kim Cascone is an experimental American composer, sound designer, and scholar who is best known as the assistant music editor for David Lynch's *Twin Peaks and Wild At Heart*, and for the ambient and industrial music released on his label, Silent Records. His current interests are laptop performance, and the use of digital glitches and systemic failure within his performance practice [11].

Nakamura and Cascone's collective interest in "indeterminacy" or "failure" within their respective performance systems is clearly demonstrated on their duo improvisation record *Color Quanta*, which was released by Silent Records in 2016 [48]. This is record characterized by its extreme frequency range, audible digital artifacts, intense glitch material, and warmly glowing analog noise. It is clear that Nakamura's and Cascone's systems are challenging them to actively navigate "systemic failure" as a means of generating novel and spontaneous improvised music [11]. Explosive bursts of noise cut to harmonious drones or sparsely spacious sonic landscapes. Cascone's harsh digital artifacts waver in and out as Nakamura's analog warmth saturates the frequency spectrum. The record sounds as if Nakamura or Cascone are constantly struggling to work their way around these failures to produce a formally consistent and sonically rich product. *Heretic's Musical Synthesis* module takes inspiration from Nakamura and Cascone's aesthetic of "indeterminacy" and "failure," by implementing stochastic processes, "mutual subversion", no-input mixer samples, and aspects of Cascone's glitch based laptop performance practice [5, 11].

2.4 Looking Forward

The historical, theoretical, and aesthetic concepts detailed in this section are the basis for *Heretic's* computational implementation. In developing *Heretic*, it was imperative to understand the history of Live Algorithms and how they are developed, the theoretical principals of free improvisation, and to completely formulate an approach to my own improvisational methodology.

3 Implementation

This chapter details how the concepts discussed in Chapter 2 are computationally implemented within *Heretic's Interpretive Listening, Contextual Decision Making*, and *Musical Synthesis* modules. This chapter also details how these modules work together to create a cohesive computer music system that produces music that adheres to my intended sonic aesthetic. The *Interpretive Listening* module consists of audio feature extraction and ten triple-layer perceptron neural-networks that are organized according to my interpretation of Anthony Braxton's *Language Music* System (Figure 2.3), enabling *Heretic* to effectively organize the sounds of my performance into a high-level organization framework. In a series of cascading Markov models, the *Contextual Decision Making* module uses Joe Morris'

Postures of Interaction in combination with data from the *Interpretive Listening module* to determine which *language type Heretic* will use in a given musical context. The *Contextual Decision Making* also determines how *Heretic* temporally actuates its sonic response, and which *structural function* will be applied to a given sound object [70]. The *Musical Synthesis* module uses data from the *Contextual Decision Making* module's Markov models to formulate a musical output via live processing, digital synthesis, and sample manipulation. These three modules are combined through interdependent communication, resulting in idiosyncratic and novel musical interactions.

3.1 System Overview

Heretic employs a modular design in which each module contains sub-module components that collectively form a composite module (Figure 3.1). The *Interpretive Listening* module contains a feature extraction sub-module in SuperCollider and a *language type* classifier sub-module using artificial neural-networks in the machine learning software Wekinator [26,77]. The *Interpretive Listening* module also employs sub-modules that smooth the data from the *language type* classifiers and track this smoothed *language type* data to determine the performer's incoming *language type*. This enables *Heretic* to interpret a human performer's real-time playing within the context of my *language music* system.

The *Contextual Decision Making* module consists of four cascading 2nd order Markov chains. The first Markov chain in this series generates *Heretic's* internal behavioral state via Joe Morris' *postures of interaction*. This determines how *Heretic* interacts with the performer's incoming *language type* data from the *Interpretive Listening* module [44]. Then using this incoming *language type* from the neural-networks and the decided internal behavioral state, the *Contextual Decision Making* module employs three other Markov models to decide the specific *language type Heretic* will implement, and which of Smalley's *onset* and *termination* structural functions will be applied to a particular *language type*

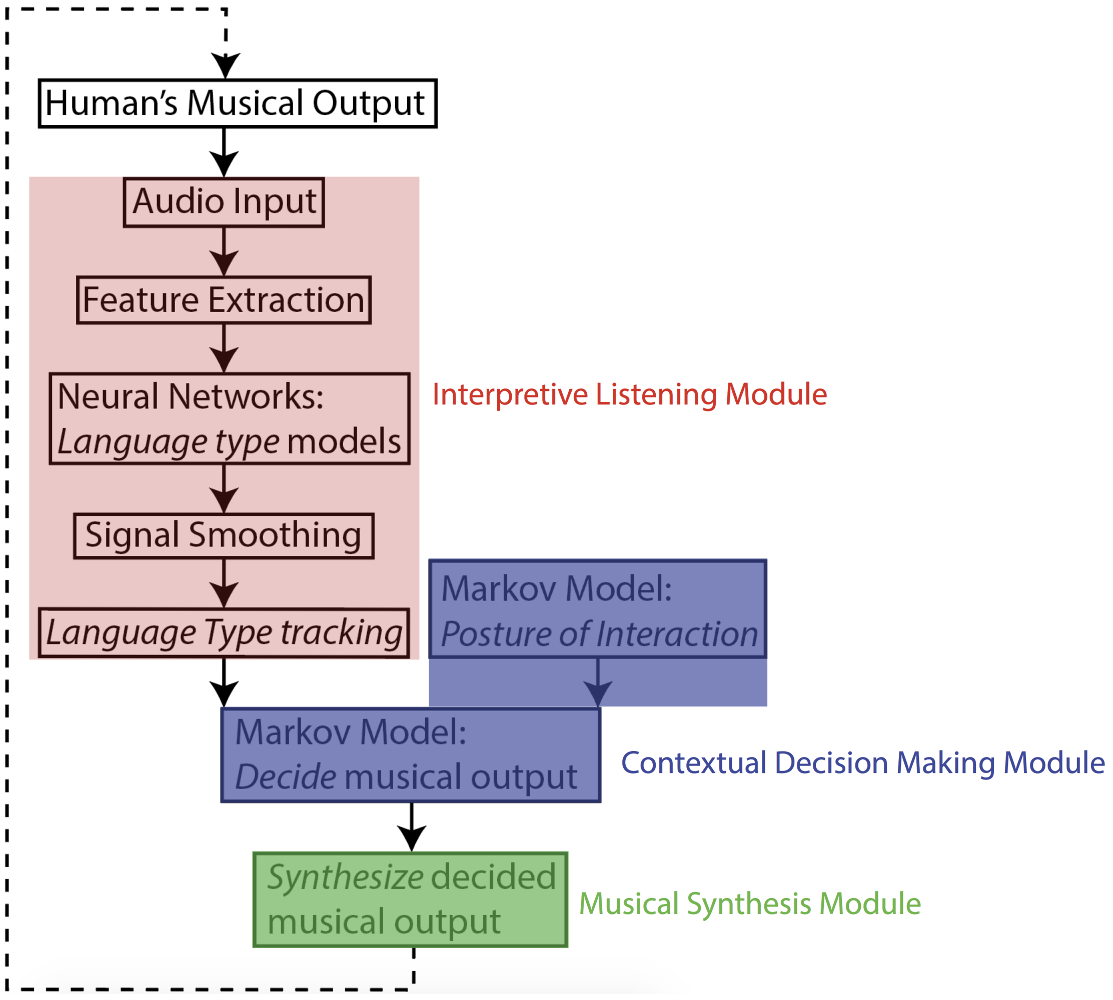

Human's Musical Output
Audio Input
Feature Extraction
Neural Networks: Language type models
Signal Smoothing
Language Type tracking
Interpretive Listening Module
Markov Model: Posture of Interaction
Markov Model: Decide musical output
Contextual Decision Making Module
Synthesize decided musical output
Musical Synthesis Module

Figure 3.1: *Heretic's* complete system architecture, displaying all of its interconnected modules and sub-modules.

utterance [70]. Once *Heretic* formulates its musical output, *Heretic* synthesizes this output via a collection of live processing and/or audio synthesis modules. The dashed line in Figure 3.1 symbolizes the bi-directional relationship or *emotional transduction* between *Heretic* and its human partner [37]. In other words, the human performer must consider *Heretic's* output when making musical decisions and vice versa, enabling a *common language* to emerge [5].

3.2 Interpretive Listening Module

Heretic's Interpretive Listening module functionality consists of three sub-modules: feature extraction, machine learning via neural-networks, and *language type* tracking. The feature extraction module in SuperCollider acts as *Heretic's* listening apparatus, parsing amplitude, timbre, and note onset audio features from a real-time audio signal [13]. These audio features are then routed to a collection of ten neural-networks in Wekinator via OSC [26]. Collectively, these neural-networks serve as a interpretive function, classifying the audio features into a *language type* model. Before *Heretic's* intended real-time application, the neural-network collection is trained on features extracted from audio samples that are representative of my complete *language type* collection (Figure 2.3). This offline approach to training *Heretic's* neural-networks encodes the *Interpretive Listening* module with prior musical knowledge; enabling *Heretic* to organize musical materials in to a high-level structure in real-time (Section 2.2.1). In computational terms, this neural-network training stage enables *Heretic* to classify low-level audio features into a definable *language type* model during an improvisation. Once the neural-networks are trained on these *language type* audio samples, *Heretic* is ready to interpret what it hears during an improvisation. Audio features are extracted from a real-time audio signal, and are routed into the neural-networks' inputs. The neural-networks categorizes audio features from the improviser's audio signal into a *language type* model, and outputs real-time percentages of how closely my playing fits each *language type* model. This detected *language type* produced by the output layer is then routed to the *Contextual Decision Making* module for a variety of creative mappings within SuperCollider.

3.2.1 Feature Extraction

The audio features used to train *Heretic's Interpretive Listening* module depend on the intrinsic sonic characteristics of each *language type*. Based on my reinterpreted version

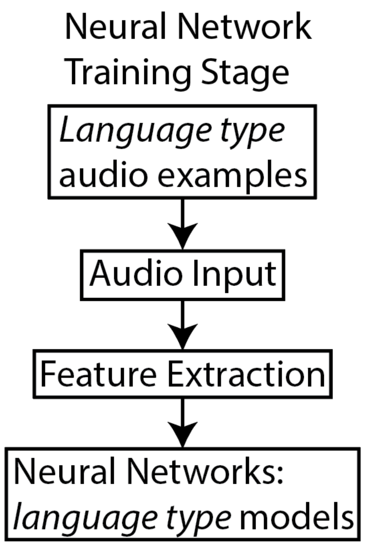
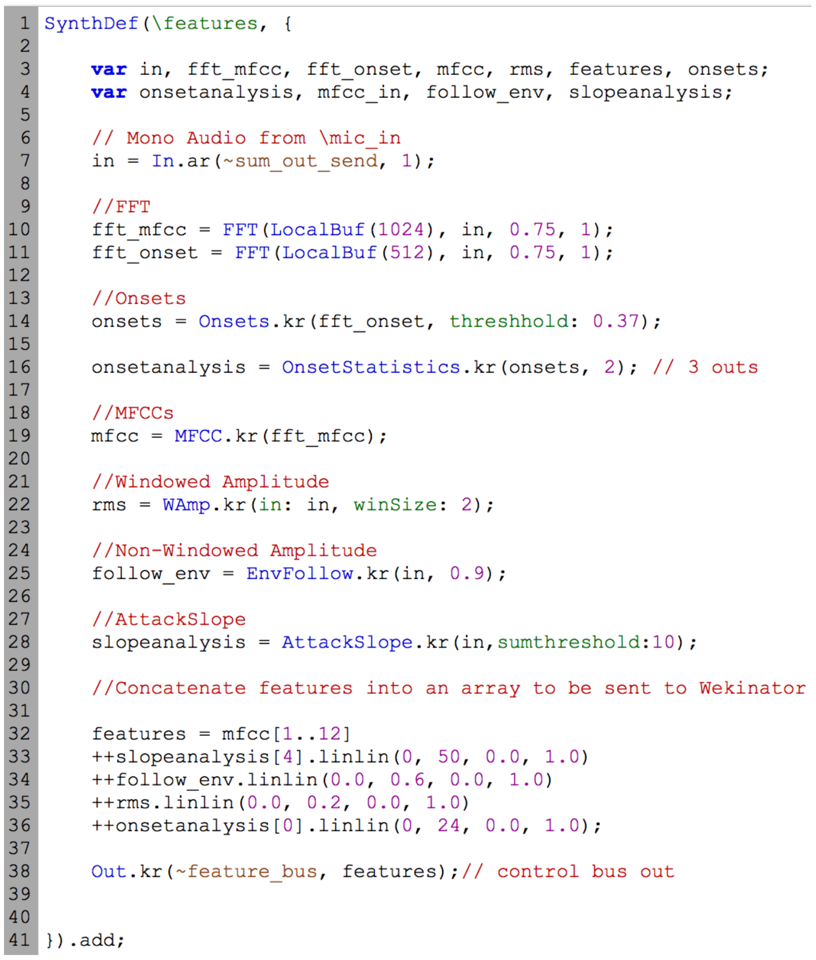

Neural Network
Training Stage
Language type audio examples
Audio Input
Feature Extraction
Neural Networks: language type models

Figure 3.2: *Heretic's* neural-network training stage.

of Braxton's language music system, I have determined windowed note onset density, inter-onset interval variance, Mel-Frequency Ceptrum Coefficents (MFCCs), windowed root-mean-square (RMS) data, amplitude envelop tracking, and attack slope data to provide an accurate analysis of my *language types*. These audio features are used to train *Heretic's* neural-networks to classify my playing into a *language type* model. This section provides justification, and examples of why each particular audio feature is useful in defining my *language type* collection.

Windowed Note Onset Density

Time encoded audio features are crucial for accurately describing *language types* that are defined by how they move through time. A windowed onset density detector keeps track of how many onsets occurred in a past window [22]. For example, windowed note onset density is effective in determining the difference between *Silence* and *Sparse Formings*, because there are moments within *Sparse Forming* that contain silence. However, if silence is currently present (no onsets), but onsets are present in the past window, then *Heretic* can interpret what its hearing as a *Sparse Forming* as opposed to *Silence*. If there were no onsets in the past window *Heretic* will interpret my playing as *Silence*.



```
SynthDef(\features, {

    var in, fft_mfcc, fft_onset, mfcc, rms, features, onsets;
    var onsetanalysis, mfcc_in, follow_env, slopeanalysis;

    // Mono Audio from \mic_in
    in = In.ar(~sum_out_send, 1);

    //FFT
    fft_mfcc = FFT(LocalBuf(1024), in, 0.75, 1);
    fft_onset = FFT(LocalBuf(512), in, 0.75, 1);

    //Onsets
    onsets = Onsets.kr(fft_onset, threshhold: 0.37);

    onsetanalysis = OnsetStatistics.kr(onsets, 2); // 3 outs

    //MFCCs
    mfcc = MFCC.kr(fft_mfcc);

    //Windowed Amplitude
    rms = WAmp.kr(in: in, winSize: 2);

    //Non-Windowed Amplitude
    follow_env = EnvFollow.kr(in, 0.9);

    //AttackSlope
    slopeanalysis = AttackSlope.kr(in,sumthreshold:10);

    //Concatenate features into an array to be sent to Wekinator

    features = mfcc[1..12]
    ++slopeanalysis[4].linlin(0, 50, 0.0, 1.0)
    ++follow_env.linlin(0.0, 0.6, 0.0, 1.0)
    ++rms.linlin(0.0, 0.2, 0.0, 1.0)
    ++onsetanalysis[0].linlin(0, 24, 0.0, 1.0);

    Out.kr(~feature_bus, features);// control bus out


}).add;
```

Figure 3.3: *Heretic's* feature extraction implementation in SuperCollider.

Inter-Onset Interval Variance

The temporal variance between onsets in an analysis window enables *Heretic* to determine whether or not I am playing a steady tempo [65]. The *Pulse Forming* is defined by rhythmic regularity and a steady tempo. When the inter-onset interval variance outputs a low number, the detected onsets are regularly consistent, telling *Heretic* that a *Pulse Forming* is my current *language type*. If the inter-onset interval variance outputs a high number, then the intervals between each onset are rapidly changing, therefore my current *language type* cannot be a *Pulse Forming*.

Mel-Frequency Ceptrum Coefficents

An array of thirteen MFCCs are at the heart of *Heretic's* Interpretive Listening module, highlighting timbre's important role in my improvisational *language types*. MFCCs are particularly apt at classifying unpitched sounds by their timbral profiles [29, 40]. For example, since *Drone Formings* and *Transgressive Formings* are capable of containing the same dynamic level, onset density, and onset variance, MFCCs are able to differentiate between the pitched timbre of a *Drone Forming* and the noisey timbre of a *Transgressive Forming*. Another use of the MFCC is to differentiate between the use of soft mallets in *Melodic Formings* and the use of drum sticks in *Polyrhythmic Formings*.

Windowed RMS and Amplitude Envelop Following

$$output = present - windowsize \tag{3.1}$$

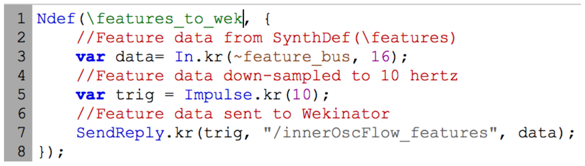
The windowed RMS feature extracts the RMS from an incoming signal while calculating this RMS value's moving average over a window of arbitrary duration [80] (Equation 3.1). The amplitude envelope follower enables *Heretic* to track sharp changes in dynamic level [19, 31]. These features accurately measure how a signal's dynamic level changes over time in relation to its current dynamic level. For example, MFCCs might easily confuse *Bombastic Formings* and *Sporadic Formings*, because there are moments within a *Sporadic Forming* that are timbrally similar to *Bombastic Formings*. Since *Sporadic Formings* often cut to silence, using a windowed RMS follower to track how a signal's amplitude changes over time enables the neural-networks to recognize a *Sporadic Forming*, instead of a *Bombastic Forming* followed by brief periods of *Silence*.

Attack Slope

The attack slope feature enables *Heretic* to differentiate between the spectral density of a note's immediate onset [54]. One example of this feature's usage is for *Heretic* to quickly classify a *Sparse Forming* versus a *Sporadic Forming*. This is because *Sporadic Formings* have a denser spectral profile upon its immediate onset than a *Sparse Forming*. Relying solely on amplitude following to classify a *Sparse Forming* versus a *Sporadic Forming* would fail, because it is possible a note onset within a *Sparse Forming* could have a greater amplitude than an onset within a *Sporadic Forming*. The attack slope also assists in the immediate classification of a *Sparse Forming*, because the windowed onset density feature's windowing function adds latency to this classification task.

Classifying Features

The above subsections describe how this combination of audio features work together to enable *Heretic* to *listen* to my improvisational language, and to *interpret* these improvisational utterances into definable *language type* models in real-time via a neural-network machine learning method. Once the optimal features for describing my collection of *language types* (Figure 2.3) have been selected and implemented in SuperCollider's feature extraction functionality, these features are concatenated into a large array (Figure 3.3), and normalized to avoid undesired weighting within the neural-networks. The array is then down-sampled to the control rate, and routed into Wekinator's neural-networks via OSC for training (Figure 3.4).



```
Ndef(\features_to_wek, {
    //Feature data from SynthDef(\features)
    var data= In.kr(~feature_bus, 16);
    //Feature data down-sampled to 10 hertz
    var trig = Impulse.kr(10);
    //Feature data sent to Wekinator
    SendReply.kr(trig, "/innerOscFlow_features", data);
});
```

Figure 3.4: Down-sampled feature extraction data sent to the neural-networks in Wekinator.

3.2.2 Neural-Networks

Neural-networks have been used for many computational tasks within computer music research. Sarroff and Casey have used auto-encoded artificial neural-networks in real-time audio synthesis systems [64]. Others have used neural-networks specifically for musical representation and style imitation as seen in Todd, Mozer, and Soukup's "connectionist music composition" and Zukowski and Carr's "Generating Black Metal and Math Rock" [45, 46, 75, 85]. In terms of representing improvisational methodologies, Eck and Schmidhuber used Long Short Term Memory (LSTM) recurrent neural-networks in their generative blues improvisation system [23]. Some creators of Live Algorithms use neural-networks to implement various organizational structures within their Live Algorithms (Section 2.1.1), such as Bown and Lexer's implementation of Genetic Algorithms using Continuous-Time Recurrent neural-networks, and Young's implementation of Boulez's chord multiplication techniques in his *prosthesis* systems [8, 83].

The abstract and general nature of my *language type* classifiers makes neural-networks a good choice for this machine learning task. This is because of neural-networks' ability to classify data which has never been presented to the network before, their ability to classify data in general categories, and their ability to act as non-linear function approximators [58, 68]. For example, one *Pulse Forming* might exist at a much different tempo and utilize different timbres than another *Pulse Forming*, or one *Drone Forming* might have widely varying spectral make-up or dynamic contour as compared to another. It is also likely that I will improvise new variations of each *language type* in a live performance, and I want *Heretic* to properly classify these previously unheard musical spaces. Neural-networks are also aptly suited to learn the non-linear relationships between the audio features used in training *Heretic* and each of my *language types*.

Testing K-nearest neighbor classification, decision tree classification, linear regression, polynomial regression, and neural-networks determined the choice to use the latter. When testing these machine learning methods, each of the ten *language type* classification models were trained on ten three-minute pre-recorded audio samples that are representative of each *language type*. Next, I improvised using each *language type* as a musical prompt, and routed the real-time audio features extracted from these improvisations into each machine learning algorithm. Notes were taken of how quickly and accurately each machine learning method classified the audio into a *language type* model. After this series of tests, the neural-networks were able to more accurately classify these previously unseen improvisations into the *language type* models than the other machine learning methods.

Neural-Network Architecture

Heretic's neural-network topology uses ten binary classifiers, discriminatively-trained on each *language type* [76]. Each neural-network is a discrete model of each *language type*. Using discriminatively-trained binary classifiers is effective for three reasons: they require fewer training examples, fewer hidden layers, and less processing power to effectively train the neural-networks to execute this musical task [43, 49, 76]. As an example of how discriminatively-trained binary classifiers are implemented within *Heretic*, in training the collection of neural-networks to detect *Melodic Formings*, the improviser would set the "melodic" neural-network's output layer to "1", and the output layer of every other neural-network to "0," and then play an example of *Melodic Formings* into the neural-networks. Once each neural-network is trained and an improvised audio signal is routed to the neural-networks for analysis, the neural-networks output real-time percentages of how closely this previously unseen improvised audio fits each *language type* model. Using multiple neural-networks for *language type* detection as opposed to a single neural-network allows for multiple *language types* to be detected simultaneously, becoming in Braxton's words a "synthesis" of *language types* [39, 81].

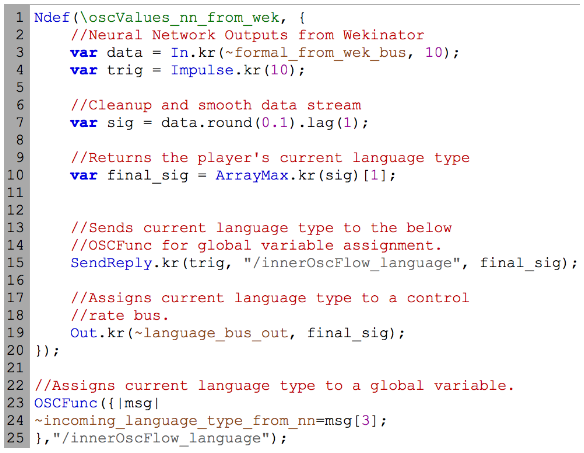
Neural-Network Hyperparameters

Once the specific neural-network topology was determined, another experiment was performed to find the optimal settings for the neural-networks' number of hidden layers. Wekinator's perceptron neural-networks include a selection of hidden layers between zero and three [26]. Since my *language types* describe abstract representations of musical activity, and lack any linear relationship between a given audio feature and a *language type*, tests using no hidden layers or one hidden layer demonstrated poor performance in the general classification of previously untrained data. This is also because single hidden layer neural-networks require significantly larger amounts of training data to solve nonlinear relationships, thus becoming more computationally expensive [58, 68]. Two hidden layers were more accurate at classifying *language types*, based on my ground truth labels, however three hidden layers showed the most accuracy in classifying my improvisations into the *language type* models.

Further neural-network hyperparameter tests focused on the number of nodes per hidden layer in order to further optimize the neural-network's performance. Using common neural-network heuristics for estimating the number of nodes per hidden layer as a starting point, I began by using the number of input nodes as the number of nodes per hidden layer, which was seventeen [58, 71]. This setting over-fit the training data, as I had to play extremely similar material to the training audio in order for the neural-network to accurately classify the *language types*. I continued this empirical process by subtracting one node per hidden layer until the neural-network exhibited the desired behavior. The optimal number of nodes per hidden layer is thirteen nodes.

Neural-Network Training and Output

After optimizing *Heretic's* feature extraction and neural-network functionality, the neural-network collection was trained on a large set of audio feature data corresponding to each *language type* model. This provided *Heretic* with the ability to accurately hear and contextualize unforeseen musical materials. As I discover new improvisational material in my personal practice sessions, I train the *language type* models on these new sonic materials as they fit within my *language type* collections. This provides the neural-networks with the flexibility to classify my future improvisational utterances. Once this extensive neural-network training stage is complete, the neural-networks are able to take the extracted features from a real-time improvisation into their input layer and output a continuously updating array of ten *language type* propability signals. Each signal is a real-time percentage of how much my playing fits each *language type* model. This array is then routed to the *Interpretive Listening* module's signal smoothing and *language type tracking* sub-modules for the purpose of detecting my incoming *language type*.



```
Ndef(\oscValues_nn_from_wek, {
    //Neural Network Outputs from Wekinator
    var data = In.kr(~formal_from_wek_bus, 10);
    var trig = Impulse.kr(10);

    //Cleanup and smooth data stream
    var sig = data.round(0.1).lag(1);

    //Returns the player's current language type
    var final_sig = ArrayMax.kr(sig)[1];


    //Sends current language type to the below
    //OSCFunc for global variable assignment.
    SendReply.kr(trig, "/innerOscFlow_language", final_sig);

    //Assigns current language type to a control
    //rate bus.
    Out.kr(~language_bus_out, final_sig);
});

//Assigns current language type to a global variable.
OSCFunc({|msg|
~incoming_language_type_from_nn=msg[3];
},"/innerOscFlow_language");
```

Figure 3.5: *Language Type* tracking in SuperCollider with data smoothing.

3.2.3 Signal Smoothing and *language type* Tracking

Before routing the neural-network signals to the *Contextual Decision Making* module, the *Interpretive Listening* module implements signal smoothing and *language type* tracking sub-modules to detect my incoming *language type* from the array of neural-network data (Figure 3.5). The signal smoothing sub-module uses decimal point rounding and a one second lag to smooth the neural-network's output. This ten-channel array of continuous, smoothed data is then sent to the *language type* tracking sub-module in SuperCollider. This sub-module uses SuperCollider's "ArrayMax" unit generator to detect the largest value in an array of data, thus returning the neural-network's decided *language type* [77]. This detected *language type* from the *language type* tracking sub-module is then routed to *Heretic's Contextual Decision Making* module.

3.3 Contextual Decision Making Module

The *Contextual Decision Making* module receives the neural-network data in SuperCollider from Wekinator via OSC, and uses this data to make a decision based on the real-time context of the improvisation. This module contains four sub-modules, each consisting of a Markov model designed to complete the following tasks:

- decide *Heretic's* internal behavioral state via Joe Morris's *postures of interaction*
- decide which *language type Heretic* will use to interact with the human's playing
- decide which of Smalley's *structural functions* will be applied to a given *language type* utterance

This section gives a brief background on Markov models as a tool within music composition, an overview of why I have chosen to use Markov models, and detailed descriptions of how I have designed each Markov model to fit my aesthetic goals for *Heretic's* musical decision making process.

3.3.1 Markov chains: An Overview

The use of Markov models in music composition dates back to Hiller and Issacson's 1956 work *Suite III for String Quartet* [30]. In 1959, Xenakis used Markov models in his compositions *Analogique A* for string orchestra, *Analogique B* for sinusoidal sounds, and *Syrmos* for 18 strings [82]. Xenakis used many simultaneous Markov chains to generate these compositions [3]. Each Markov chain generates a different musical parameter such as frequency, amplitude, density, or length. Specific combinations of these parameters constitutes a "screen," and parametrically moving between "screens" defines the formal structure of a piece. Thus, Xenakis is crafting a work's low-level material via the Markov models, and is then generating the work's formal structure from these stochatically generated low-level materials. Since Xenakis, many other composers have used Markov chains for various tasks in music composition, such as David Cope's melodic sequence modelling in his *Alice* system, and Charles Dodge's melodic and rhythmic modelling of Stephen Foster songs [17, 21].

More recent implementations of Markov chains in the domain of computer music have involved the use of improvisation. For example, David Zicarelli's *Jam Factory* system enables performers to change the weights of Markov chains in real-time to generate novel pitch sequences [84]. Sertan Senturk's machine learning system uses a database of 77 Turkish Folks songs to train Variable-Length Markov models to generate improvised pitch sequences based on the Turkish Folk Song Style [67]. Christopher Dobrian's improvisation software implements a version of Xenakis' "screens" by enabling the real-time saving of a screen's parametric state, and enabling interpolation between these states [20]. Assayag and Dubnov use statistically trained Variable-Length Hidden Markov models to implement *factor oracles* within *Omax* [4].

3.3.2 Markov Chains: *Heretic*

The choice to use Markov chains in *Heretic's Contextual Decision Making* module stems from Derek Bailey's concept of *mutual subversion* [5]. Mutual subversion is when a common musical language between two or more improvisers is so collectively well-known, that the musicians are able to unpredictably test the confines of this language by challenging the musical decisions of other players. While *Heretic* can hear my language and interpret its contextual meaning in real-time, a Markov chain's stochastic properties allow *Heretic* to artfully subvert the human's playing while also testing the limits of the machine and human's shared musical language. In other words, a Markov chain's probabilistic weights might suggest to the human performer what decision to expect from *Heretic*. However, the stochastic properties of a Markov chain can easily subvert these expectations, leading to the human performer constantly being faced with how to navigate through these confirmed or subverted expectations.

MorrisMarkov

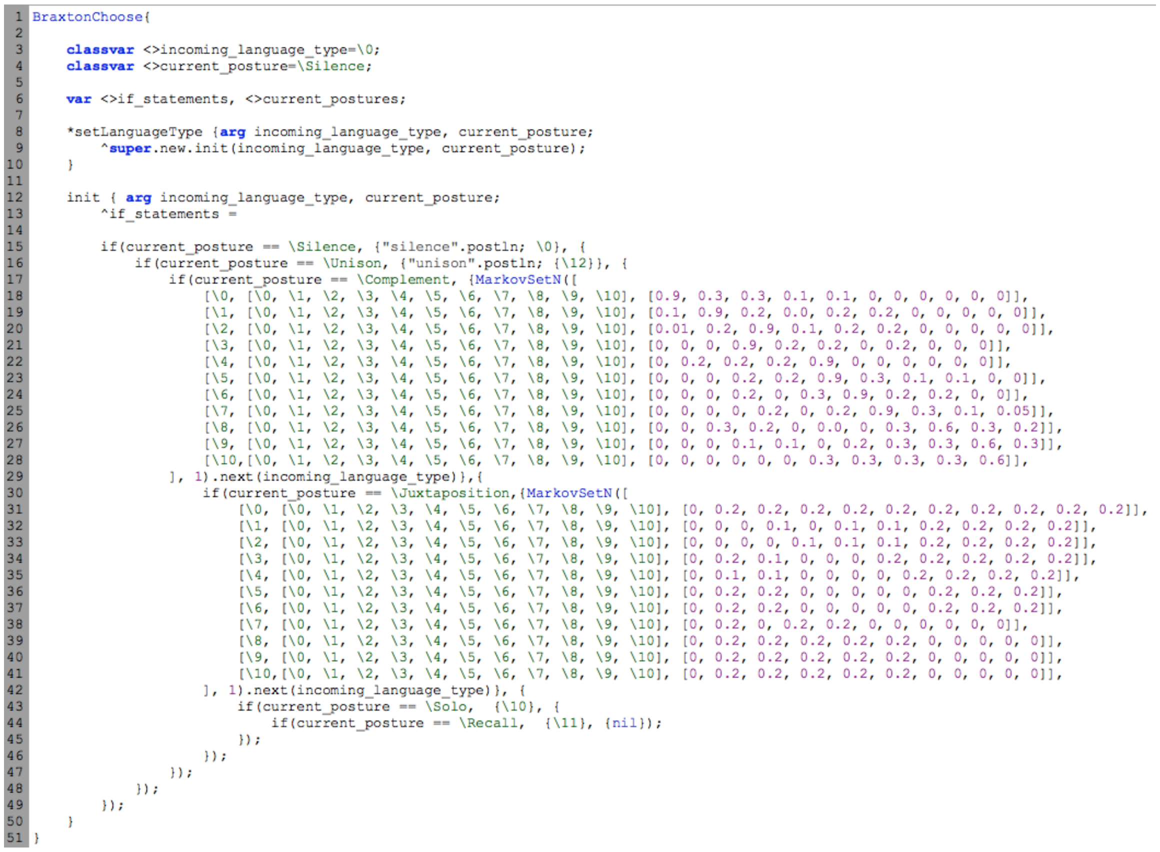
The first Markov chain in this cascading series, MorrisMarkov, is a second-order Markov chain that decides which *posture of interaction Heretic* will implement within a musical situation. In addition to Morris' *posture of interaction*, I have included a *Recall* posture of interaction within this Markov chain. The *Recall* posture retrieves and actuates a synthesis module that *Heretic* used earlier in the performance. This functionality enables reoccurring sonic motifs to take shape, an improvisational technique that is part of my approach to creating cohesive formal structures during an improvisation.

Before a performance, *Heretic's* user sets the probabilistic weights for each *posture of interaction*, thus generating a type of mood or disposition regarding how *Heretic* behaves during the improvisation. For example, if the *Unison posture of interaction* weight is set high while the others are set to zero, *Heretic* will simply follow the human performer's sonic choices throughout the performance. A contrasting example, if the *Solo* posture of interaction weight is set high while the others are set to zero, *Heretic* bypasses the *Interpretive Listening* module, thus completely ignoring the human's musical input.

With the resultant musical output as the motivating force, MorrisMarkov's weighting paradigm contains two constraints. First, since MorrisMarkov is a second-order Markov chain, it uses the current and past postures of interaction to decide the next posture of interaction. If the same posture has been chosen twice in a row, a hard-coded weight of zero restricts that posture from repeating a third time (Figure 3.6). The second constraint is that the *Recall* posture cannot be called until two postures have been previously chosen in the performance. This is because not enough music has taken place for it to make sense for *Heretic* to recall a previously used musical state.

BraxtonChoose

The BraxtonChoose Markov model uses the decided posture of interaction from Morris-Markov and the human performer's incoming *language type* from the *Interpretive Listening*



```
BraxtonChoose{

    classvar <>incoming_language_type=\0;
    classvar <>current_posture=\Silence;

    var <>if_statements, <>current_postures;

    *setLanguageType {arg incoming_language_type, current_posture;
        ^super.new.init(incoming_language_type, current_posture);
    }

    init { arg incoming_language_type, current_posture;
        ^if_statements =

        if(current_posture == \Silence, {"silence".postln; \0}, {
            if(current_posture == \Unison, {"unison".postln; {\12}}, {
                if(current_posture == \Complement, {MarkovSetN([
                    [\0, [\0, \1, \2, \3, \4, \5, \6, \7, \8, \9, \10], [0.9, 0.3, 0.3, 0.1, 0.1, 0, 0, 0, 0, 0, 0]],
                    [\1, [\0, \1, \2, \3, \4, \5, \6, \7, \8, \9, \10], [0.1, 0.9, 0.2, 0.0, 0.2, 0.2, 0, 0, 0, 0, 0]],
                    [\2, [\0, \1, \2, \3, \4, \5, \6, \7, \8, \9, \10], [0.01, 0.2, 0.9, 0.1, 0.2, 0.2, 0, 0, 0, 0, 0]],
                    [\3, [\0, \1, \2, \3, \4, \5, \6, \7, \8, \9, \10], [0, 0, 0, 0.9, 0.2, 0.2, 0, 0.2, 0, 0, 0]],
                    [\4, [\0, \1, \2, \3, \4, \5, \6, \7, \8, \9, \10], [0, 0.2, 0.2, 0.2, 0.9, 0, 0, 0, 0, 0, 0]],
                    [\5, [\0, \1, \2, \3, \4, \5, \6, \7, \8, \9, \10], [0, 0, 0, 0.2, 0.2, 0.9, 0.3, 0.1, 0.1, 0, 0]],
                    [\6, [\0, \1, \2, \3, \4, \5, \6, \7, \8, \9, \10], [0, 0, 0, 0.2, 0, 0.3, 0.9, 0.2, 0.2, 0, 0]],
                    [\7, [\0, \1, \2, \3, \4, \5, \6, \7, \8, \9, \10], [0, 0, 0, 0, 0.2, 0, 0.2, 0.9, 0.3, 0.1, 0.05]],
                    [\8, [\0, \1, \2, \3, \4, \5, \6, \7, \8, \9, \10], [0, 0, 0.3, 0.2, 0, 0.0, 0, 0.3, 0.6, 0.3, 0.2]],
                    [\9, [\0, \1, \2, \3, \4, \5, \6, \7, \8, \9, \10], [0, 0, 0, 0.1, 0.1, 0, 0.2, 0.3, 0.3, 0.6, 0.3]],
                    [\10,[\0, \1, \2, \3, \4, \5, \6, \7, \8, \9, \10], [0, 0, 0, 0, 0, 0, 0.3, 0.3, 0.3, 0.3, 0.6]],
                ], 1).next(incoming_language_type)},{
                    if(current_posture == \Juxtaposition,{MarkovSetN([
                        [\0, [\0, \1, \2, \3, \4, \5, \6, \7, \8, \9, \10], [0, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2]],
                        [\1, [\0, \1, \2, \3, \4, \5, \6, \7, \8, \9, \10], [0, 0, 0, 0.1, 0, 0.1, 0.1, 0.2, 0.2, 0.2, 0.2]],
                        [\2, [\0, \1, \2, \3, \4, \5, \6, \7, \8, \9, \10], [0, 0, 0, 0, 0.1, 0.1, 0.1, 0.2, 0.2, 0.2, 0.2]],
                        [\3, [\0, \1, \2, \3, \4, \5, \6, \7, \8, \9, \10], [0, 0.2, 0.1, 0, 0, 0, 0.2, 0.2, 0.2, 0.2, 0.2]],
                        [\4, [\0, \1, \2, \3, \4, \5, \6, \7, \8, \9, \10], [0, 0.1, 0.1, 0, 0, 0, 0, 0.2, 0.2, 0.2, 0.2]],
                        [\5, [\0, \1, \2, \3, \4, \5, \6, \7, \8, \9, \10], [0, 0.2, 0.2, 0, 0, 0, 0, 0, 0.2, 0.2, 0.2]],
                        [\6, [\0, \1, \2, \3, \4, \5, \6, \7, \8, \9, \10], [0, 0.2, 0.2, 0, 0, 0, 0, 0, 0.2, 0.2, 0.2]],
                        [\7, [\0, \1, \2, \3, \4, \5, \6, \7, \8, \9, \10], [0, 0.2, 0, 0.2, 0.2, 0, 0, 0, 0, 0, 0]],
                        [\8, [\0, \1, \2, \3, \4, \5, \6, \7, \8, \9, \10], [0, 0.2, 0.2, 0.2, 0.2, 0.2, 0, 0, 0, 0, 0]],
                        [\9, [\0, \1, \2, \3, \4, \5, \6, \7, \8, \9, \10], [0, 0.2, 0.2, 0.2, 0.2, 0.2, 0, 0, 0, 0, 0]],
                        [\10,[\0, \1, \2, \3, \4, \5, \6, \7, \8, \9, \10], [0, 0.2, 0.2, 0.2, 0.2, 0.2, 0, 0, 0, 0, 0]],
                    ], 1).next(incoming_language_type)}, {
                        if(current_posture == \Solo,  {\10}, {
                            if(current_posture == \Recall,  {\11}, {nil});
                        });
                    });
                });
            });
        });
    }
}
```

Figure 3.6: The BraxtonChoose Markov model with *if statement* constraints implemented as a class in SuperCollider.

module to decide which *language type Heretic* will use as its musical output. These chosen *language types* manifest themselves as *synth instances* in *Heretic's Musical Synthesis* module (Section 3.4). BraxtonChoose is implemented as a class that uses if statement constraints to decide which synthesis bank in the *Musical Synthesis* module to activate. As seen in Figure 3.6, if *Heretic's* "current posture" is *Silence*, then *Heretic's* BraxtonChoose class returns the symbol "0," the *Silence* synthesis bank is activated, terminating all current synthesis tasks (henceforth, refer to Table 3.1 for the symbol representation of *Heretic's* synthesis banks within SuperCollider).

Regarding the *Unison* posture of interaction, I have interpreted this posture to act as a fusion of my sound with *Heretic's* sound. Therefore, the *Unison* synthesis banks are separate from the other banks, because they only contain live processing modules that act as an extension of my drum sound rather than as a separate musical voice. There are ten *Unison* synthesis banks, each specifically designed for each *language type*. When MorrisMarkov returns a *Solo* or *Recall* posture, the "if" statements constraints in BraxtonChoose will simply activate an algorithmic generated *Solo* passage or *Recall* a previously synth synth instance.

Symbol	*language type*
0	Silence
1	Sparse Formings
2	Drone Formings
3	Granular Formings
4	Melodic Formings
5	Pulse Formings
6	Polyrhythmic Formings
7	Sporadic Formings
8	Transgressive Formings
9	Bombastic Formings
10	Solo Posture
11	Recall Previous Synth
12	Unison Posture

Table 3.1: Symbol representation of *Heretic's language type* synthesis banks in SuperCollider.

```
this_synth = currentEnvironment.at(("unison_bank_"++~incoming_language_type.asString).asSymbol).choose;
~recall_synth_array.add(this_synth);
```

Figure 3.7: *Heretic* using the incoming *language type* from the *Interpretive Listening* module to choose a synthesizer from a *Unison* synthesis bank. Whenever a synthesizer is called, it is added to *recall_synth_array*, enabling *Heretic* to potentially recall this synthesizer later in the improvisation.

The *Complement* and *Juxtaposition* postures use the human performer's incoming *language type* to determine how it will complement or juxtapose this *language type*. The probabilistic weights are set based on my aesthetic reasoning behind which *language types* complement or juxtapose one another. For example, I have distributed weights for a *Complement* posture with an incoming *language type* of *Pulse Formings* in the order of most likely to be chosen to less likely to be chosen: *Pulse Formings*, *Polyrhythmic Formings*, *Melodic Formings*, *Granular Formings*, *Sporadic Formings*, and *Transgressive Formings* (Figure 3.6). In contrast, the distributed weights for *Juxtaposition* posture with an incoming *language type* of *Pulse Formings* are equal across the following *language type* banks, *Sparse Formings*, *Drone Formings*, *Sporadic Formings*, *Transgressive Formings*, and *Bombastic Formings* (Figure 3.6), with all of the other *language type* weights set to zero. This weighting process is repeated for all of the other *language types* with aesthetic consideration in mind. During a complement posture, if BraxtonChoose returns the same *language type* as the human performer's incoming *language type*, this is not considered a *Unison* posture, because these synthsizers act as separate musical voices as opposed to the *Unison* live processing modules (Section 3.4).

Temporal Actuation

Heretic must now decide when and how to implement these synth instances. Once a performance begins, a random synth instance is actuated. From this point forward, each new synth instance is actuated once the previous synth instance is terminated, with temporal variances applied via the use delays to avoid unwanted temporal regularity. A synth instance's temporal actuation and termination changes depending on *Heretic's* current posture of interaction. For example, if *Heretic's* current posture of interaction is *Unison* or *Complement*, the synth instance is actuated and terminated in close proximity to when the human performer changes *language types* (Figure 3.8). A synth associated with a *Complement* posture has the added probability of a slight delay in its onset or termination, because a complementary interaction does not suggest as tight of a one-to-one temporal mapping as a *Unison* interaction. During *Solo* or *Juxtaposition* postures, *Heretic* autonomously chooses how long a synth instance will last and the precise actuation and termination of the said synth instance. The combination of tight temporal mapping with a the human performer and *Heretic's* flexible temporal actuation enables a varied and nuanced temporal structure to emerge.

Before the improvisation begins, the human performer selects the length of the improvisation and enters this time into *Heretic's* graphical user interface. When the performance begins, *Heretic* actuates a clock that tracks the improvisation's length. Once the clock reaches the predetermined end time, depending on which *posture of interaction Heretic* is currently in, *Heretic* proceeds to decide how to end the improvisation with the human performer. If *Heretic* is in a *Unison* or *Complement* posture of interaction, it will end its output as soon the human performer goes silent. If *Heretic* is in a *Solo* or *Juxtaposition* posture of interaction, it will end its output once its current synth routine ends, which could be a minute or two after the previously set ending time.

```
e.add(\unison_sequence-> {
    currentEnvironment.put(("synth_routine_"++~synth_counter.asString).asSymbol,
        Routine({
            var synth_num = ~synth_counter, this_langauge_type = ~incoming_language_type;
            ~synth_trig[\12].value; //Triggers "Unison" Synth adhering to the human performer's current language type.
            loop{
                //As soon as the human performer changes language type, the synth instance is terminated.
                if(~incoming_language_type != this_langauge_type,
                    {
                        Routine({
                            currentEnvironment.at(("synth_instance_"++synth_num.asString).asSymbol).set(\gate, 0);
                            e[\synths_sequence_choose].value; //Triggers next synth instance upon terminating this synth.
                            s.sync;
                            currentEnvironment.at(("synth_routine_"++synth_num.asString).asSymbol).stop;
                        }).play(SystemClock);

                }, {nil});

                0.5.wait;
            };
        }).play(SystemClock);
    );
});
```

Figure 3.8: *Unison* synthesis temporal actuation routine.

```
e.add(\juxtaposition_sequence-> {
    currentEnvironment.put(("synth_routine_"++~synth_counter.asString).asSymbol,
        Routine({
            var synth_num = ~synth_counter;
            ~synth_trig[BraxtonChoose.setLanguageType(~incoming_language_type, ~current_posture)].value;
            ~choose_solo_wait.choose.wait;
            currentEnvironment.at(("synth_instance_"++synth_num.asString).asSymbol).set(\gate, 0);
            s.sync;
            e[\synths_sequence_choose].value
        }).play(SystemClock);
    );
});
```

Figure 3.9: *Juxtaposition* synthesis temporal actuation routine.

SmalleyMarkovOnset and SmalleyMarkovTerminate

Once the BraxtonChoose Markov model decides which *language type Heretic* will activate in its musical response and when, the SmalleyMarkovOnset and SmalleyMarkovTerminate Markov chains decide which of Denis Smalley's "onset" and "termination" structural functions will be applied to a given sound object [70]. In a literal sense, the "onset" and "termination" structural functions act as the attack envelop and release envelop segments of

Heretic's chosen synthesised *language type* output [70] (Figure 2.1). The envelope device used to generate each structural function is SuperCollider's "dadsr," a standard adsr envelope with an added delay argument (Figure 3.10). Each envelope's segments are chosen from arrays generated during *Heretic's* initialization phase. These arrays are designed according to my interpretation of Smalley's *structural functions* while avoiding any overt regularity or predictability in the shape and timing of the envelopes. This is done in order to induce an affect of *mutual subversion* between *Heretic* and the human performer.

I have hand-selected the following "onset" structural functions from Smalley's list: *attack, emergence, anacrusis*, and *upbeat* (Figure 2.1). I have also added a *delay* option to the choice of possible onsets. The delay onset simply delays the attack segment of one of the other onset functions by an arbitrary number of seconds. Considering each new synth instance is actuated as soon as the previous one is terminated, this *delay* onset allows for a sort of musical breath between synth instance actuations. The probabilistic weights of each onset function are determined by which onsets make the most musical sense in regard to each specific *language type*. For example, if *Heretic* chooses a *Drone Forming*, it is most likely that SmalleyMarkovOnset will apply an *emergence* onset function, considering the slow temporal nature of a *Drone Forming*. If a "delay" onset function were chosen, a delay time would be chosen from an array of possible delay times, and SmalleyMarkovOnset would activate its Markov chain again with the constraint of not having the "delay" function as an option. Another example is if *Heretic* chooses a *Sporadic Forming*, only an "attack" onset function will be used to enhance this *language type's* initial burst of onset energy, and a "closure" termination function accentuates this *language type*'s quick and disjunct sonic character.

Like SmalleyMarkovOnset, I have hand-selected a specific group of his "terminations" and one borrowed from Smalley's list of "continuants." The terminations used in this Markov chain are *arrival, disappearance, release, closure*, and from the continuants list, *prolongation*. The *prolongation* acts analogously to the *delay* onset function from SmalleyMarkovOnset. When *Heretic* calls for a synth instance to be terminated, if SmalleyMarkovTerminate chooses *prolongation*, SmalleyMarkovTerminate is accessed again without the option of *prolongation* and simply adds a delay to when the termination function will be applied to the synth instance. This *prolongation* function enables multiple synth instances to overlap, adding to the potential sonic complexity and intended macro-temporal irregularity of a given musical event. SmalleyMarkovTerminate's probabilistic weighting functions in a similar manner to SmalleyMarkovOnset, with each weight determined by mapping specific termination probabilities to a *language type's* sonic nature. For example, SmalleyMarkovTerminate is most likely to choose a *disappearance* termination because it enhances a *Drone Forming's* slow-moving nature.

```
Env.dadsr(~delay_time.choose, ~emergence_time.choose, 0.01, 1, ~dissappearance_time.choose, 1
    ,[~atk_cur.choose, ~dec_cur.choose, ~sus_cur.choose, ~rel_cur.choose]);
```

Figure 3.10: *Heretic's* envelope functionality using SuperColliders "dadsr" (delay-attack-decay-sustain-release) envelop generator. This is an example of a delayed emergence onset with a disappearance termination.

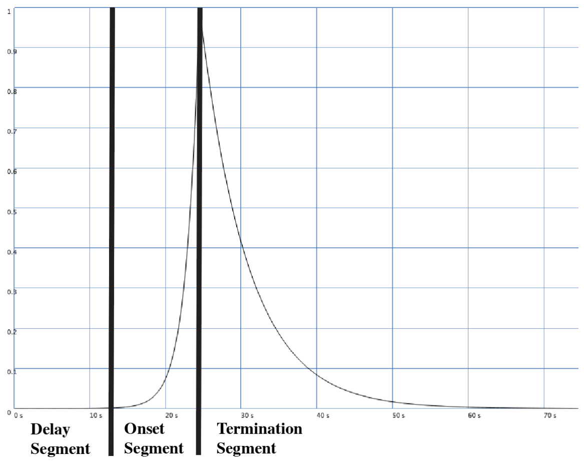


Figure 3.11: An example of a *structural function* with a delay, onset, and termination segment.

Regarding Smalley's *Continuants*

The "continuant" segment of each synth instance is determined by the intrinsic sonic nature of the synth as opposed to an externally prescribed enveloping function. For example, if a synth uses low-level audio feature data from the human performer as a control signal, the quality of the synth's continuation will be determined by whatever the human performer plays. This particular example could be interpreted as a *passage* continuant. Another example is if *Heretic* applies a *disappearance* termination to a *Drone Forming*, while simultaneously actuating a *Granular Forming* with an *emergence* onset, this could be perceived as a *transition* continuation between these two musical states.

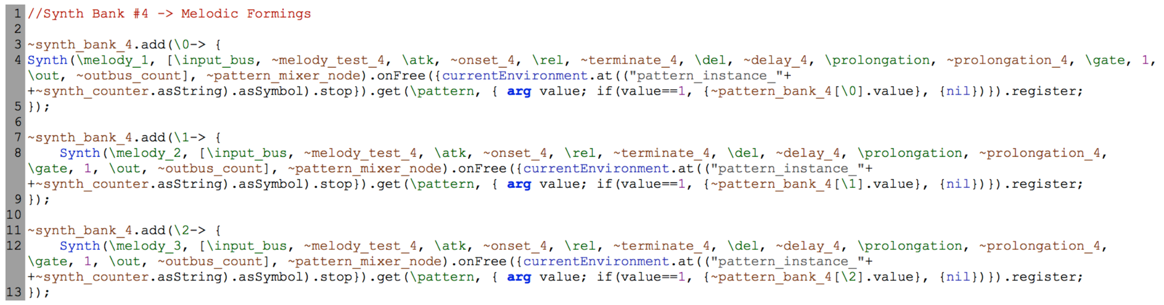
3.3.3 Decisions Manifested as Music

The decision making processes described in this section determine, **how** *Heretic* will interact with a human performer, **what** *language type* it will implement within this interaction, **when** it will actuate this *language type*, and **where** in the structural framework of the improvisation it will implement a particular synth instance. Once all of these decisions have been made, *Heretic* uses its sonic voice to express these decisions. The following section details *Heretic's* low-level audio synthesis framework and the particular aesthetic intentions embedded in this framework.

3.4 Musical Synthesis Module

Heretic's Musical Synthesis module is a collection of functionalities that express *Heretic's* musical decision making process as music. Henceforth, I will refer to these sonic functionalities as individual *synth instances*. *Heretic's* synth instances are implemented using a combination of SuperCollider's SynthDef and Pattern functionalities [78]. A synth instance incorporates the the macro-level decisions made by the BraxtonChoose Markov model, and the strucual functions decided by the SmalleyMarkovOnset and SmalleyMarkovTerminate Markov models. The micro-level decisions for each synth instance are determined by a synth's sonic parameters, which are generated algorithmically or via low-level audio features extracted from the human's playing. Each synth instance is classified into *Heretic's* own version of Braxton's *Language Music*, enabling a "common language" and "musical coherence" between *Heretic* and myself to emerge [5,62]. This combination of macro and micro level organization within *Heretic's Musical Synthesis* module forms *Heretic's* own "sound," or unique musical voice to take shape [5,44].

3.4.1 Organization and Synth Instance Selection



```
//Synth Bank #4 -> Melodic Formings

~synth_bank_4.add(\0-> {
Synth(\melody_1, [\input_bus, ~melody_test_4, \atk, ~onset_4, \rel, ~terminate_4, \del, ~delay_4, \prolongation, ~prolongation_4, \gate, 1,
\out, ~outbus_count], ~pattern_mixer_node).onFree({currentEnvironment.at(("pattern_instance_"+
+~synth_counter.asString).asSymbol).stop}).get(\pattern, { arg value; if(value==1, {~pattern_bank_4[\0].value}, {nil})}).register;
});

~synth_bank_4.add(\1-> {
    Synth(\melody_2, [\input_bus, ~melody_test_4, \atk, ~onset_4, \rel, ~terminate_4, \del, ~delay_4, \prolongation, ~prolongation_4,
\gate, 1, \out, ~outbus_count], ~pattern_mixer_node).onFree({currentEnvironment.at(("pattern_instance_"+
+~synth_counter.asString).asSymbol).stop}).get(\pattern, { arg value; if(value==1, {~pattern_bank_4[\1].value}, {nil})}).register;
});

~synth_bank_4.add(\2-> {
    Synth(\melody_3, [\input_bus, ~melody_test_4, \atk, ~onset_4, \rel, ~terminate_4, \del, ~delay_4, \prolongation, ~prolongation_4,
\gate, 1, \out, ~outbus_count], ~pattern_mixer_node).onFree({currentEnvironment.at(("pattern_instance_"+
+~synth_counter.asString).asSymbol).stop}).get(\pattern, { arg value; if(value==1, {~pattern_bank_4[\2].value}, {nil})}).register;
});
```

Figure 3.12: This code demonstrates adding a *Melodic Forming* synth instance to the *Melodic Forming language type* synth dictionary. Here, the sound sources are generated algorithmically via SuperCollider's Pattern functionality. The audio output of these Patterns are routed through an envelope which applies a *structural function* to the audio output of said Pattern instance.

Heretic's musical language is organized via my reinterpreted version of Braxton's *Language Types* [9,39] (Figure 2.3). Each of *Heretic's language types* act as a organizational group that houses a variety of synth instances that fit the sonic properties of each *language type*. In SuperCollider, a dictionary contains individual synth instances that are grouped according to their specific *language type* characteristics. When the BraxtonChoose Markov model returns *Heretic's* decided *language type*, *Heretic* freely chooses a synth instance at random from the synth bank dictionary that corresponds to the returned *language type*. While *Heretic* is maintaining "musical coherence" by intelligently choosing a *language type* at a macro-level via its *Interpretive Listening* module, it also maintains "mutual subversion" by randomly deciding the specific synth instance it will implement at a given moment [5,62].

3.4.2 SynthDefs and Patterns

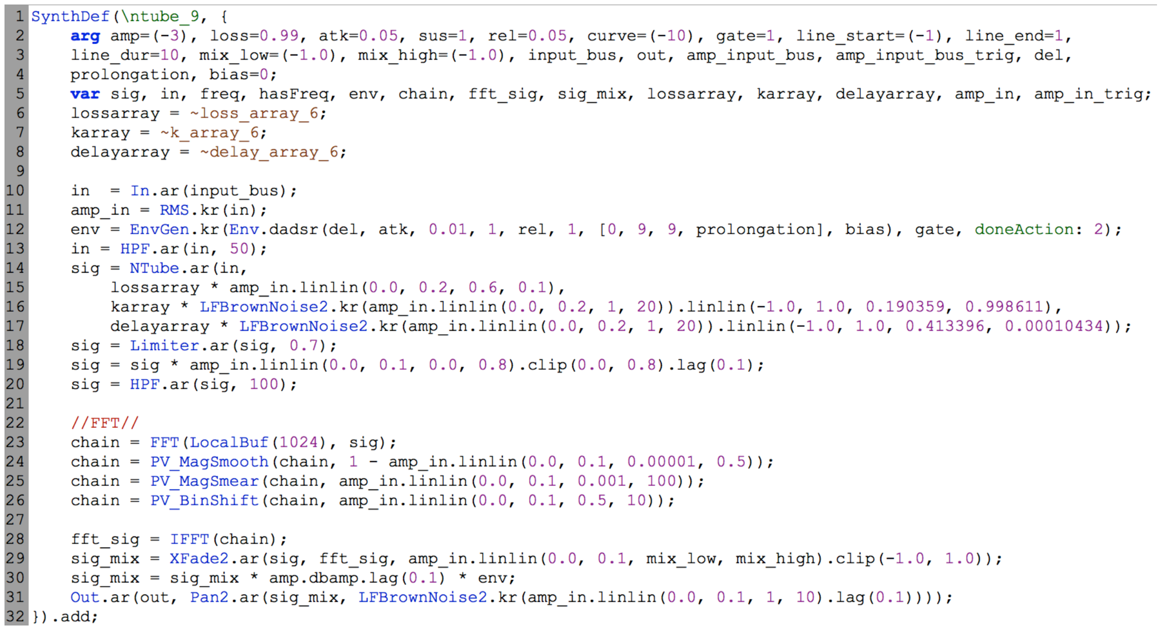
Heretic's Musical Synthesis module uses two of SuperCollider's audio synthesis functionalities, SynthDefs and Patterns [77]. In my experience, SynthDefs and Patterns are useful for different musical tasks. SynthDefs are useful for processing acoustic sounds in real-time, and mapping low-level audio features to synthesized sound. Patterns excel in creating algorithmically generated sequences of music, sampling pre-recorded sound, and generating streams of data to control SynthDefs. Using SuperCollider's flexible audio and data routing functionalities, I have developed a method for using SynthDefs and Patterns in unification with each other.

SynthDefs

SynthDefs are a flexible and efficient way of generating, manipulating, and processing audio in Supercollider. As stated in the Supercollider Documentation:

"Once the server has a synth definition, it can very efficiently use it to make a number of synths based on it. Synths on the server are basically just things that make or process sound, or produce control signals to drive other synths" [42].

As previously detailed, the *Unison* synth dictionaries only apply live processing effects to the acoustic signal coming from the human performer. Figure 3.13 is a SynthDef that uses an audio effect developed by Nick Collins entitled "NTube.ar" [15]. NTube.ar processes real-time audio using a a physical model of a tube with N tube sections, N-1 scattering junctions in between each tube section, and a variable array of delay lines [15]. My implementation of this effect in a SynthDef uses real-time RMS data from the *Interpretive Listening* module and maps this data onto various synthesis parameters (Section 3.2.1) (Figure 3.13). Mapping this RMS data to such a large number of parameters results in a chaotic sound, making this SynthDef a *Unison Bombastic Forming*. By transforming and extending the drum-set's sonic properties using RMS tracking, *Heretic's* output and the human performer's output morph into one electronic/acoustic hybrid, a true sonic *Unison*.



```
SynthDef(\ntube_9, {
    arg amp=(-3), loss=0.99, atk=0.05, sus=1, rel=0.05, curve=(-10), gate=1, line_start=(-1), line_end=1,
    line_dur=10, mix_low=(-1.0), mix_high=(-1.0), input_bus, out, amp_input_bus, amp_input_bus_trig, del,
    prolongation, bias=0;
    var sig, in, freq, hasFreq, env, chain, fft_sig, sig_mix, lossarray, karray, delayarray, amp_in, amp_in_trig;
    lossarray = ~loss_array_6;
    karray = ~k_array_6;
    delayarray = ~delay_array_6;

    in  = In.ar(input_bus);
    amp_in = RMS.kr(in);
    env = EnvGen.kr(Env.dadsr(del, atk, 0.01, 1, rel, 1, [0, 9, 9, prolongation], bias), gate, doneAction: 2);
    in = HPF.ar(in, 50);
    sig = NTube.ar(in,
        lossarray * amp_in.linlin(0.0, 0.2, 0.6, 0.1),
        karray * LFBrownNoise2.kr(amp_in.linlin(0.0, 0.2, 1, 20)).linlin(-1.0, 1.0, 0.190359, 0.998611),
        delayarray * LFBrownNoise2.kr(amp_in.linlin(0.0, 0.2, 1, 20)).linlin(-1.0, 1.0, 0.413396, 0.00010434));
    sig = Limiter.ar(sig, 0.7);
    sig = sig * amp_in.linlin(0.0, 0.1, 0.0, 0.8).clip(0.0, 0.8).lag(0.1);
    sig = HPF.ar(sig, 100);

    //FFT//
    chain = FFT(LocalBuf(1024), sig);
    chain = PV_MagSmooth(chain, 1 - amp_in.linlin(0.0, 0.1, 0.00001, 0.5));
    chain = PV_MagSmear(chain, amp_in.linlin(0.0, 0.1, 0.001, 100));
    chain = PV_BinShift(chain, amp_in.linlin(0.0, 0.1, 0.5, 10));

    fft_sig = IFFT(chain);
    sig_mix = XFade2.ar(sig, fft_sig, amp_in.linlin(0.0, 0.1, mix_low, mix_high).clip(-1.0, 1.0));
    sig_mix = sig_mix * amp.dbamp.lag(0.1) * env;
    Out.ar(out, Pan2.ar(sig_mix, LFBrownNoise2.kr(amp_in.linlin(0.0, 0.1, 1, 10).lag(0.1))));
}).add;
```

Figure 3.13: The Ntube SynthDef that extends and morphs the the real-time drum-set sound by mapping RMS data to various synthesis parameters. The RMS data in this SynthDef is represented by the variable "amp_in."

SynthDefs can also generate sound via the low-level audio features extracted from the human performer's playing. An example of this is the "concat_delay" SynthDef. This SynthDef can be categorized in many different *language type* synth banks, due to its ability to mirror the human performer's playing in real-time, and its ability to access a wide variety of audio corpuses. "Concat_delay" uses zero crossing rate, log mean square amplitude, spectral centroid, and spectral tilt features to retrieve and concatenate audio samples from any stored audio corpus that matches the real-time features extracted from the acoustic signal [12,66]. Therefore, when its output is routed through a delay line, *Heretic* mirrors the phrasing and gesture of the incoming audio at irregular times while avoiding a tight one to one mapping.

Patterns

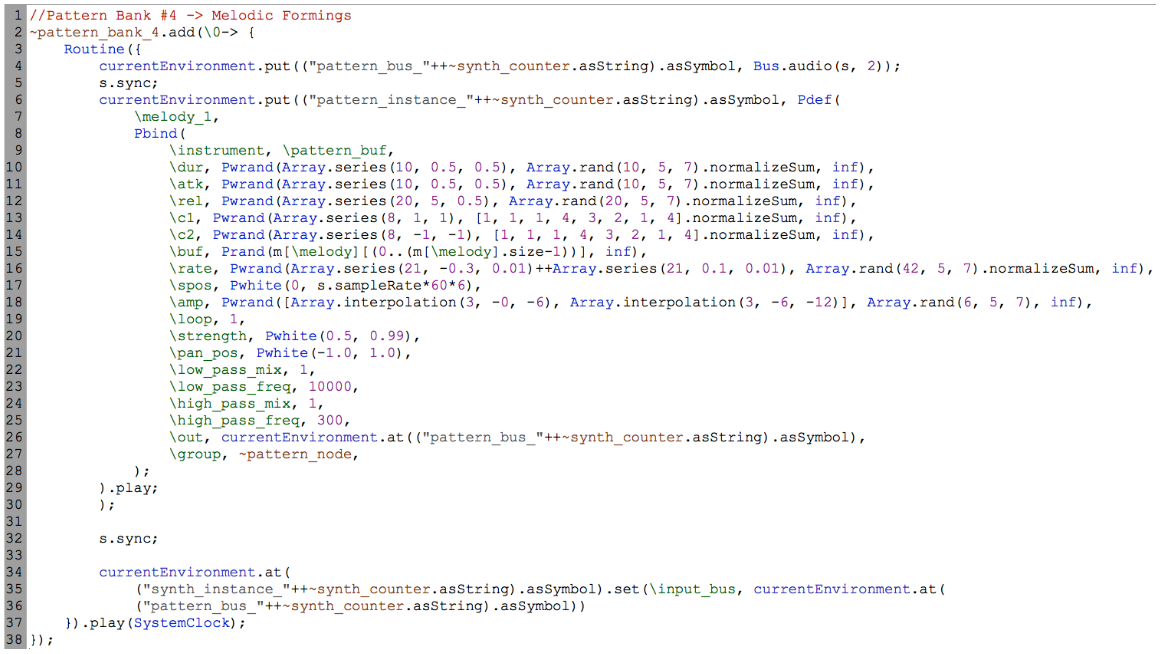
Heretic uses Patterns to create algorithmically generated sequence of music. While Patterns do not literally generate sound themselves, they enable high-level control of many musical parameters within a SynthDef. James Harkins supplies a eloquent explanation of Pattern's functionality in his online article *A Practical Guide to Patterns:*

"The SuperCollider Pattern library... is a higher-level representation of a computational task... when they are appropriate, they free the user from worrying about every detail of the process. Using patterns, one writes what is supposed to happen, rather than how to accomplish it. In SuperCollider, patterns are best for tasks that need to produce sequences, or streams of information" [28].

In my particular implementation of Patterns within *Heretic*, the sonic output of a Pattern is routed through a SynthDef that applies the structural functions determined by SmalleyMarkovOnset and SmalleyMarkovTerminate to the composite amplitude envelope of the Pattern's output. This is done in order for the same structural functions to be consistently applied to both SynthDefs and Patterns.

Figure 3.14 shows an example of a Pattern in the *Melodic Formings* synth bank array. This pattern strings together a collection of small melodic motifs that were prerecorded using analog synthesizers. This Pattern accesses these samples through a simple SynthDef designed to playback and manipulate audio. The resulting sound consists of multiple meandering melodies that are generated autonomously via stochastic processes.

The example in Figure 3.15 uses stochastic processes to generate "clouds" of various synthesized drum sounds followed by a variable amount of silence before actuating another "cloud" of drum sounds [82]. This *Sporadic Forming* adheres to Bailey's concept of "mutual subversion" because the amount of silence between each cloud constantly changes, keeping the human performer guessing when *Heretic* might actuate another cloud of drum sounds.



```
//Pattern Bank #4 -> Melodic Formings
~pattern_bank_4.add(\0-> {
    Routine({
        currentEnvironment.put(("pattern_bus_"++~synth_counter.asString).asSymbol, Bus.audio(s, 2));
        s.sync;
        currentEnvironment.put(("pattern_instance_"++~synth_counter.asString).asSymbol, Pdef(
            \melody_1,
            Pbind(
                \instrument, \pattern_buf,
                \dur, Pwrand(Array.series(10, 0.5, 0.5), Array.rand(10, 5, 7).normalizeSum, inf),
                \atk, Pwrand(Array.series(10, 0.5, 0.5), Array.rand(10, 5, 7).normalizeSum, inf),
                \rel, Pwrand(Array.series(20, 5, 0.5), Array.rand(20, 5, 7).normalizeSum, inf),
                \c1, Pwrand(Array.series(8, 1, 1), [1, 1, 1, 4, 3, 2, 1, 4].normalizeSum, inf),
                \c2, Pwrand(Array.series(8, -1, -1), [1, 1, 1, 4, 3, 2, 1, 4].normalizeSum, inf),
                \buf, Prand(m[\melody][(0..(m[\melody].size-1))], inf),
                \rate, Pwrand(Array.series(21, -0.3, 0.01)++Array.series(21, 0.1, 0.01), Array.rand(42, 5, 7).normalizeSum, inf),
                \spos, Pwhite(0, s.sampleRate*60*6),
                \amp, Pwrand([Array.interpolation(3, -0, -6), Array.interpolation(3, -6, -12)], Array.rand(6, 5, 7), inf),
                \loop, 1,
                \strength, Pwhite(0.5, 0.99),
                \pan_pos, Pwhite(-1.0, 1.0),
                \low_pass_mix, 1,
                \low_pass_freq, 10000,
                \high_pass_mix, 1,
                \high_pass_freq, 300,
                \out, currentEnvironment.at(("pattern_bus_"++~synth_counter.asString).asSymbol),
                \group, ~pattern_node,
            );
        ).play;
        );

        s.sync;

        currentEnvironment.at(
            ("synth_instance_"++~synth_counter.asString).asSymbol).set(\input_bus, currentEnvironment.at(
            ("pattern_bus_"++~synth_counter.asString).asSymbol))
    }).play(SystemClock);
});
```

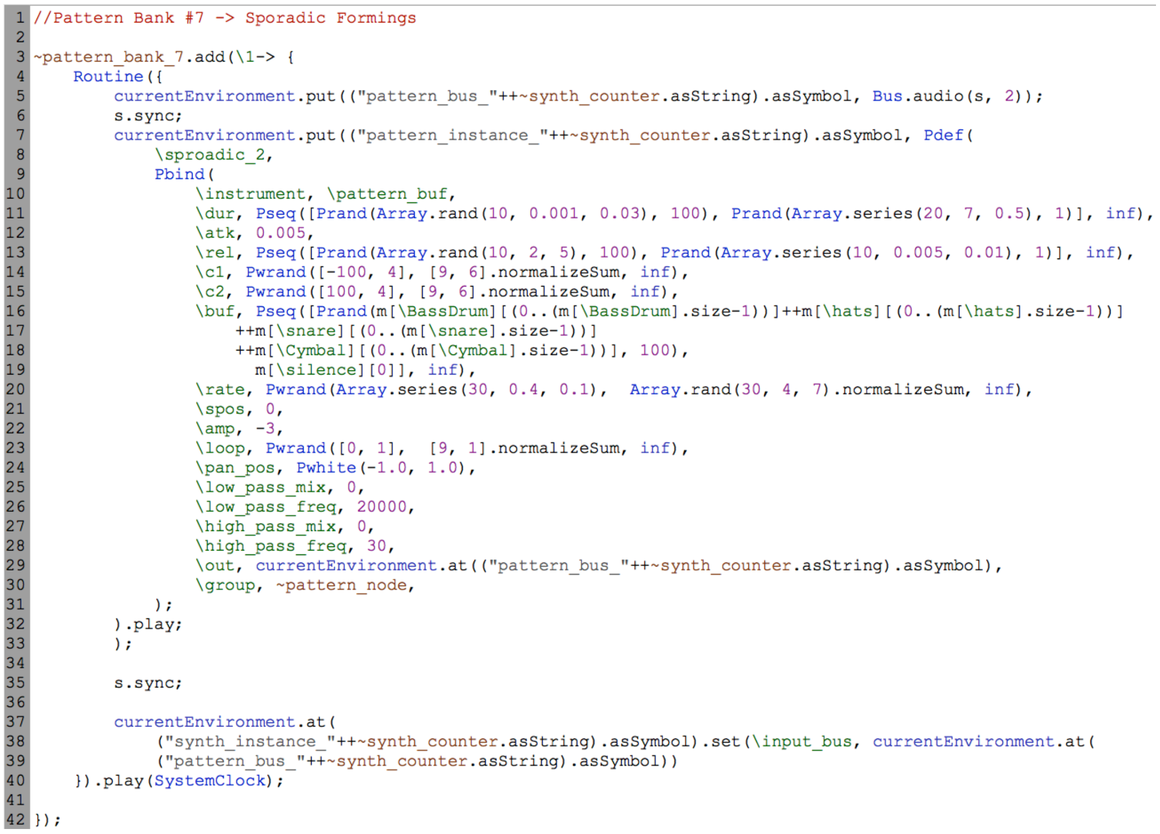
Figure 3.14: An example of a *Melodic Forming* Pattern.

3.4.3 Sonic Aesthetics: Creating a "Sound"

At the core of *Heretic's* musical language are the literal sounds it produces. In his book, *The Proprieties of Free Music*, Joe Morris discusses the importance of sonic identity and originality when performing freely improvised music:

"Originality (invention) is manifested in a sound, which is quickly associated with the performer or group of performers who make it... the goal is always an identifiable sound that contains the artist's audible and inaudible creative, expressive DNA" [44].

Just like a human improviser, it is important for the creators of machine improvisers and Live Algorithms to imbue their systems with a "sound," or an "expressive DNA" [44]. Considering *Heretic* is using my improvisational methodology as a basis for enabling its own musical voice to take shape, *Heretic's* sonic voice is a synthesis of various sonic properties from musical idioms and specific improvisers that have inspired my own musical practice as a composer and improviser (Section 2.3). As discussed in section 3.4.2, *Heretic* uses live



```
//Pattern Bank #7 -> Sporadic Formings

~pattern_bank_7.add(\1-> {
    Routine({
        currentEnvironment.put(("pattern_bus_"++~synth_counter.asString).asSymbol, Bus.audio(s, 2));
        s.sync;
        currentEnvironment.put(("pattern_instance_"++~synth_counter.asString).asSymbol, Pdef(
            \sproadic_2,
            Pbind(
                \instrument, \pattern_buf,
                \dur, Pseq([Prand(Array.rand(10, 0.001, 0.03), 100), Prand(Array.series(20, 7, 0.5), 1)], inf),
                \atk, 0.005,
                \rel, Pseq([Prand(Array.rand(10, 2, 5), 100), Prand(Array.series(10, 0.005, 0.01), 1)], inf),
                \c1, Pwrand([-100, 4], [9, 6].normalizeSum, inf),
                \c2, Pwrand([100, 4], [9, 6].normalizeSum, inf),
                \buf, Pseq([Prand(m[\BassDrum][(0..(m[\BassDrum].size-1))]++m[\hats][(0..(m[\hats].size-1))]
                    ++m[\snare][(0..(m[\snare].size-1))]
                    ++m[\Cymbal][(0..(m[\Cymbal].size-1))], 100),
                      m[\silence][0]], inf),
                \rate, Pwrand(Array.series(30, 0.4, 0.1),  Array.rand(30, 4, 7).normalizeSum, inf),
                \spos, 0,
                \amp, -3,
                \loop, Pwrand([0, 1],  [9, 1].normalizeSum, inf),
                \pan_pos, Pwhite(-1.0, 1.0),
                \low_pass_mix, 0,
                \low_pass_freq, 20000,
                \high_pass_mix, 0,
                \high_pass_freq, 30,
                \out, currentEnvironment.at(("pattern_bus_"++~synth_counter.asString).asSymbol),
                \group, ~pattern_node,
            );
        ).play;
        );

        s.sync;

        currentEnvironment.at(
            ("synth_instance_"++~synth_counter.asString).asSymbol).set(\input_bus, currentEnvironment.at(
            ("pattern_bus_"++~synth_counter.asString).asSymbol))
    }).play(SystemClock);

});
```

Figure 3.15: A *Sporadic Forming* Pattern that generates stochastic "clouds" of synthesized drum sounds [82].

processing effects, digital synthesis, and sample manipulation as means of expressing its musical voice. Here, I will provide examples of each of these sonic functionalities, how they represent my aesthetic tastes, and how they relate to the inspirational musics discussed in Section 2.3.

Live Processing Effects

Live processing effects extend, augment, and manipulate the real-time sound of acoustic instruments [1]. While many Live Algorithms output a singular musical voice [33, 35, 37, 50, 51], the opportunity to alter acoustic instruments with the real-time processing capabilities of a computer is an expressive musical tool to add to a Live Algorithm's sonic palette.

The duo project of Sam Pluta and Peter Evans is an excellent example of acoustic/electronic fusion via live processing. In particular, I have taken inspiration from Pluta's effectiveness in electronically processing, and subsequently fusing with Evans' acoustic trumpet sound. There are many examples of sonic funsion in the track "Dark Matter" from Pluta/Evan's record *Event Horizon*, which was released by Pluta's record label, *Carrier Records* in 2014 [56].

One example is at the beginning of the improvisation with Evans playing a phrase of complex, bass heavy growls. At 00:14, Evans begins to add dynamic accents while maintaining the drone-like growls, the timbral quality of the growls slightly shifting with each accent. Evans continues this material until approximately 00:40 as Pluta emerges from Evans' texture. At first, Pluta's entrance appears to be another shift in Evans' pitch, but 00:48 clarifies that Pluta is "freezing," and pitch shifting the spectral quality of Evans' sound. An accent similar to Evans' accents peeks out of the texture whenever Pluta shifts the freeze effect, a digital extension of Evans' trumpet. Another example of Pluta's live processing abilities being used to sonically fuse with Evan's acoustic sound occurs around 02:18 as Pluta applies a rapid delay to Evans' real-time signal. As this delay fades in, Evans mirrors the delay's elongated phrase length and pitched-up quality. This is an archetypal example of Evans reproducing electronic sounds, a crucial element to the duo's fused language.

Heretic's Musical Synthesis module implements various SynthDefs that approach live processing effects in a similar manner to these musical examples from "Dark Matter" [56]. One example is the "NTube.ar" SynthDef discussed in section 3.4.2 (Figure 3.13), which extends an acoustic sound source using the physical modeling of filtered delay lines. I have also implemented at spectral shift freeze delay effect similar to the one used by Pluta at approximately 00:40 in the recording of "Dark Matter." When actuated during a *Transgressive Forming*, this effect extends the "noise oriented" quality of an acoustic sound source by wildly shifting the spectral quality and pitch of a frozen spectral profile in real-time. This is especially effective when the acoustic sound source is bowed styrofoam, megaphone feedback, or bowed cymbals. Another example is a "SwitchDelay" effect that behaves in a similar manner to the delay effect Pluta applies to Evans' trumpet sound at

02:18 in the aforementioned recording [57]. This "SwitchDelay" effect can enhance a *Pulse Forming* when its delay time is fixed, or it can augment a *Polyrhythmic Forming* when multiple delay lines are used. These are just a few examples of many live processing effects *Heretic* may implement as tools for sonic fusion during an improvisation.

Digital Synthesis

Heretic accesses SuperCollider's wide range of digital synthesis functionalities to express its sonic voice's computer-like qualities. For example, *Heretic* implements standard additive synthesis techniques to create complex *Drone Formings*, while always applying some form of analog modelling to the drone's signal to maintain a sonic character consistent with the analog synthesizer and no-input mixer samples used in its sample manipulation functionality. SuperCollider contains various Moog filter emulators that assist in adding this analog character to digital oscillators [27, 72]. This desire to achieve a synthesis sound with an analog character is inspired by the use of analog drones in Frank Rosaly's *Centering and Displacement* [60, 61]. This album's B-side begins with a thick, saturated, low-frequency focused drone that lasts until 05:28. The drone is static enough to allow Rosaly's solo to take the forefront, yet the drone also contains inner movements that adds a subtle counter-point to Rosaly's playing. In *Heretic's Drone Formings*, the static nature of the drone is achieved via the digital oscillators, and the subtle manipulations of the drone comes from the Moog filter emulators.

An example of digital synthesis in a melodic context can be found in Pluta/Evan's "Dark Matter" [56]. Pluta begins the section at 06:55 by occasionally modulating the fundamental carrier frequency of his wobbly sine-tone up an 11th from a D5 to a G6. When Evans enters, he engages in a *unison* posture by playing rhythmic patterns on the notes G#4 and D#6. Both Pluta and Evans' note selections are compound intervals that are 4ths and 5ths when reduced to an octave. One method for creating *Melodic Formings* within *Heretic* are through Patterns that use Markov chains to generate a stream of pitches from a pitch class. The attack, decay, sustain, and release of each note generated by the Pattern can also be generated using Markov chains. When using *Transgressive Formings*, *Heretic* outputs extreme sonic utterances, such as high pitched sine tones. Inspiration for using piercing sine tones as a *Transgressive Forming* comes from the track "Spectral Light" in Nakamura and Cascone's album *Color Quanta*, in which the entire track is based on a 14,847hz sine tone. While Nakamura and Cascone's use of extreme frequencies in this track lacks sonic energy stemming from rhythmic density or loudness, this simple gesture of a soft 14,847hz sine-tone creates a substantial amount of musical tension.

Sample Manipulation

The samples used within *Heretic* vary from field recordings, acoustic instrument recordings, found object recordings, analog synthesizer recordings, no-input mixer recordings, and recordings of my previous compositions. I am continuously adding samples to this collection with the intention that *Heretic's* sonic palette is always evolving. These samples are not simply triggered by *Heretic* as that would compromise *Heretic's* musical autonomy. Instead, *Heretic* implements various Patterns that can access any sample, sequence these samples together into a novel musical gesture, and autonomously manipulate the samples via re-sampling, pitch-shifting, reversing, and sending the output of the sampler to a variety of live processing effects.

I have chosen to include analog synthesizer and no-input mixer recordings to *Heretic's* sample collection to reinforce the analog character of sound as discussed in the previous subsection. Also, the sonic quality of the no-input mixer is integral to my "sound" as a composer, as I am heavily influenced by the sonic aesthetic from Toshimaru Nakamura's no-input mixer improvisations, and have used sounds from the no-input mixer in many of my recent works [47]. The no-input mixer samples used in *Heretic* are from various recordings taken from my performances on the no-input mixer. This is similar to Frank Rosaly's approach to creating *Centering and Displacement* where her edited together his electronic and percussion improvisations into a complete piece of music (Section 2.3.2). However, the difference in this context is that *Heretic* is autonomously re-contextualizing these recorded improvisations by using them as improvisational materials via splicing, rearranging, and manipulation.

Heretic also uses recordings of my complete compositions as sampling material. This is inspired by Anthony Braxton's work *Echo Echo Mirror House*, in which a computer program designed by Carl Testa can play any recording from Braxton's complete discography [74]. Another way *Heretic* implements sonic re-contextualization is by recording its own output to a wav file and storing it on the computer's hard-disk for later use in its sampling functionality. This method enables *Heretic* to always have novel sonic materials to actuate in a performance while autonomously generating its own sonic materials.

3.4.4 A Novel Musical Voice

This section details the inner workings of *Heretic's Musical Synthesis* module and its resultant sonic aesthetic and unique musical voice. This unique musical voice results from *Heretic's* ability to interpretively listen to a human performer, make a musical decision based on what it hears, and implement this musical decision while maintaining "an identifiable sound that contains [its own] audible and inaudible creative, expressive DNA" [44]. The Live Algorithms detailed in 2.1 have their own unique voices, but these voices are restricted to a limited sonic palette. *Heretic's* "identifiable sound" differs from these other Live Algorithms by assuming the vast sonic palate of a flexible laptop improviser [55]. *Heretic* achieves this through the novel use of live processing effects, analog modelled digital synthesis, and sampling techniques that use a highly nuanced and eclectic collection of samples.

3.5 Towards a Musical Evaluation

This chapter describes the computational implementation of *Heretic's Interpretive Listening*, *Contextual Decision Making*, and *Musical Synthesis* modules and how these modules interact to create *Heretic's* novel musical output. It also details how *Heretic's* implementation adheres to my musical goals, aesthetic intentions, and improvisational methodology as detailed in Chapter 2. The following chapter provides a formal analysis of music created with *Heretic* as an attempt to evaluate its autonomy and creativity as an improviser. Following the *Analysis and Evaluation* chapter, I draw conclusions from *Heretic's* development, and provide ideas for future work that may stem from this research.

4

Analysis and Evaluation

This section provides a formal analysis of a recording made with myself and *Heretic*. This analysis also uses the recorded output from the *language type* classifiers to demonstrate *Heretic's* ability to accurately classify my real-time playing into a *language type model*, and how it makes musical decisions based on this data. The recording can be found at this URL: https://soundcloud.com/hunter-brown-music/heretic-full-1. In this analysis, I segment the formal structure of the improvisation, and identify specific musical moments that demonstrate *Heretic's* ability to achieve "autonomy, novelty, participation and leadership" [6]. This analysis also examines the aesthetic characteristics of *Heretic's* musical output and whether or not these characteristics fit my musical intentions. The purpose of this analysis is to evaluate *Heretic* objectively and subjectively. Evaluating *Heretic* objectively is inherently problematic, considering the motivation behind *Heretic* was to create an autonomous system that uses my own improvisational methodology as a computational and conceptual framework for machine improvisation (Section 1). Rather, I attempt to evaluate *Heretic's* interactions with a human performer and its resultant musical output via Blackwell et al.'s defining qualities of a Live Algorithm: "autonomy, novelty, participation and leadership," while also evaluating whether *Heretic* maintains my motivations behind its creation.

While there have been attempts to objectively evaluate algorithmic composition systems, I find it impossible to objectively quantify the aesthetic value of a Live Algorithm's musical performance [25, 32, 52, 53]. Pearce and Wiggins' paper *Towards A Framework for the Evaluation of Machine Compositions*, and Eigenfeldt et al's paper *Evaluating Musical Metacreation in a Live Performance Context* test whether an audience could tell the difference between a composition made by an algorithm or a by human composer [25, 52]. To qualify their study, Pearce and Wiggins state, "the means of evaluating the compositions generated by a machine will depend on the aims of the designer" [52]. Eigenfeldt et al. also acknowledge the role of the creator's approval by stating the importance that "the designer of the system accepts the output as artistically valid" [25]. As stated in Chapters 1 and 2 of this thesis, my aim is *not* to emulate music made by a human, but rather for *Heretic* to maintain its intrinsic computer-like qualities and to use my improvisational methodology as a means of generating its own unique musical voice that I find to be "artistically valid" [25]. Therefore, measuring *Heretic's* musical output against human-made music does not align with my musical goals. However, considering my aim is to develop a Live Algorithm that maintains autonomy through its interactions with a human performer, it is valid to evaluate *Heretic's* musical output using Blackwell et al.'s defining qualities of a Live Algorithm: "autonomy, novelty, participation and leadership" [6]. By using these qualities as evaluative benchmarks, I am able to objectively determine whether or not *Heretic* can be labelled as a Live Algorithm. Once confirming *Heretic's* status as a Live Algorithm, I can then evaluate

Heretic's musical output with aesthetic consideration, confirming my subjective musical intentions.

4.1 Formal Segmentation and Analysis

Before evaluating *Heretic's* autonomy, novelty, participation and leadership, it is necessary to identify this improvisation's formal sections, how *Heretic's Interpretive Listening* module influenced its decision making, how mine and *Heretic's* interactions generate these sections, and how we navigate through these sections. A labelled visual representation and spectrogram of this improvisation's formal structure is shown in Figure 4.1. The list below describes each labelled formal section, and how the formal structure progresses through the improvisation.

- **1:** I begin this improvisation by engaging in a *Drone Forming* by applying a transducer to a tom-tom, resulting in a sustained texture. *Heretic* interacts with this *Drone Forming* via a *Complement* posture of interaction by engaging in a *Melodic Forming*.
- **2:** In this transitional moment, I move from a *Drone Forming* to a *Melodic Forming*, mirroring *Heretic's* output from section 1. Once I enter this *Melodic Forming*, *Heretic* recognizes this as a *solo* posture by fading its output to silence.
- **3:** At the end of my melodic solo in section 2, I began a *Granular Forming* by rustling wooden shells. *Heretic* breaks its *Silence* posture by actuating a granular synthesizer, a *Complement* posture of interaction.
- **4:** I slowly move from the rustling wooden shells to bowed cymbals, a *Drone Forming*. *Heretic* maintains its previous *Granular Forming* as I change language types.
- **5:** After establishing my bowed cymbal *Drone Forming*, *Heretic* also engages in a *Drone Forming* while maintaining its previous *Granular Forming* for a few more

seconds. This collective *Drone Forming* is short lived as *Heretic* suddenly engages in a *Juxtaposition* posture of interaction in section 6. *Heretic's Drone Forming* in this section is recalled in section 11.

- **6:** While I am still engaging in a *Drone Forming*, *Heretic* engages in *Juxtaposition* posture in this section by implementing a variably delayed concatenative synthesizer that uses a corpus of synthesized drum sounds. Considering the short and sharp nature of these drum sounds, this *Juxtaposition* is stark against the previously established *Drone Formings*. Once *Heretic* actuates this concatenative synthesizer, I suddenly morph this interaction from a *Juxtaposition* to a *Unison* by abandoning my *Drone Forming* to engage in a *Sparse Forming*.

- **7:** This *Complementary Sparse Forming* continues, and as *Heretic* and I build in density, I move towards a *Pulse Forming*. Upon hearing this *Pulse Forming*, *Heretic* applies a delay effect to by drum sound. This *Unison* posture adds to this section's build in intensity. This is an example of "language type synthesis" as discussed in Section 3.2.2 [9, 39]. Once a climax is reached about halfway through this section, *Heretic* fades outs its *Sporadic Forming* and *Unison* delay effect. I follow this fade-out by reducing my note-onset density.

- **8:** Following a brief pause, this section is short drum solo in which *Heretic* goes silent.

- **9:** After *Heretic's* short lived bout of silence while I solo, *Heretic* joins my solo by applying a different delay effect on my drums and by actuating a rapid sequence of erratic synth instances. This intense moment of "mutual subversion" signals for me to end my solo, and that a new formal section has arrived [5].

- **10:** As this structural signpost of erratic synths ends section 9, there is a brief moment of silence before I begin a rhythmic hi-hat pattern. *Heretic* interprets my use of fast rolls on the hi-hat and this pattern's rhythmic nature as synthesis of *Granular* and *Pulse Formings*. Therefore, *Heretic* uses a *Unison* posture of interaction to sonically fuse with my hi-hat pattern by applying a spectral pitch-shifting effect to my real-time sound. This effect uses RMS data from my playing the control its parameters, thus mirroring my low-level phrase shape while adding a novel sonic character.

- **11:** Section 10 suddenly comes to an end after I fade my hi-hat pattern to pianissimo. Following this decrescendo I punctuate the phrase ending with a sharp cymbal attack, marking the begin of section 11. After this sharp attack, I begin a soft drum roll that leads into another *Drone Forming*. *Heretic* plays in *Complement* with this *Drone Forming* by recalling the *Drone Forming* from section 5.

- **12:** *Heretic* suddenly morphs its *Drone Forming* into a *Bombastic Forming*. I attempt to follow *Heretic's Juxtaposition* posture with a *Sporadic Forming*, but upon hearing the climax of *Heretic's Bombastic Forming*, I lowered the volume of my *Sporadic Forming* to enable *Heretic* to take a *Solo*. The improvisation ends with *Heretic* and myself fading to silence.

The above formal analysis confirms that *Heretic's* interactions with a human performer results in clear formal structures that "[emerge] from process" via my intended "bottom-up approach" to generating form in free improvisation [44, 59] (Section 2.2.1). These formal sections contain a wide variety of sonic characters, durations, and interactions, which adhere to my aesthetic intentions of sonic cohesion/indeterminacy, macro-temporal irregularity, "mutual subversion," and "musical coherence" [5, 62]. By achieving my intended aesthetic values and by *Heretic's* ability to "self-organize" during a performance with a human performer, *Heretic* reflects Blackwell et al.s' criteria for a true live algorithm: autonomy, novelty, participation and leadership [6].

4.2 Autonomy

Blackwell et al define *autonomy* as the ability for the live algorithm "to act and respond to unknowable and unforeseen inputs in ways that have not be completely prescribed" [6]. *Heretic* obtains autonomy because its neural networks are able to classify untrained audio signals into a definable language type model (Section 3.2.2), and it employs stochastic modelling to make musical decisions based on what it hears from the human performer (Section 3.3). Blackwell at al's definition of *autonomy* is similar to Bailey's concept of "mutual subversion," where "mutual subversion" is formed by improvisers responding to another improviser's language in a subversive way that is not previously prescribed or expected. This multi-layered framework for determining *Heretic's* autonomous musical output is demonstrated in section 12, where *Heretic's* sudden departure from a *Drone Forming* to a *Bombastic Forming* was not previously "prescribed," therefore challenging me as an improviser to navigate this subversion. Another example of *Heretic's* autonomy occurs in section 10, where *Heretic* responds to the "unforeseen" input of my sudden rhythmic and granular hi-hat pattern with a dynamic and smooth effect, a response that was unexpected and "not completely prescribed" [6].

4.3 Novelty

Novelty is achieved by "[avoiding] the cliched and the obvious" when "supporting, leading, or subverting" other musicians [6]. *Heretic's* primary method of "[avoiding] the cliched" is via the unique nature of its sonic voice. *Heretic* autonomously samples a large collection of my hand-crafted audio samples, full pieces of music, and re-sampled/re-processed audio from its own output. *Heretic* also uses the human performer's real-time output as sonic material, which is constantly changing due to the musical tendencies for improvising performers to avoid repeating material from performance to performance (Section 3.4). *Heretic's* flexible *Musical Synthesis* module, wide variety of sounds, and how *Heretic* implements these sounds in an infinite amount of combinations with the human performer's playing leads to a sonic voice that "avoids the cliched and the obvious" [6]. A specific example of *novelty* is in section 7, where *Heretic* uses a concatenative synthesizer through a multi-tapped delay line to mirror my *Sparse Forming* with synthesized drum sounds. By adding a real-time delay effect on my acoustic drum sounds while maintaining its delayed concatenative synthesizer, *Heretic* is supporting this build in energy while also leading this phrase to a new direction by adding a delay effect. The obvious method of supporting this phrase would have been to simply increase the feedback parameter on the concatenative synthesizer's multi-taped delay lines. *Heretic* instead adds a non-obvious real-time delay effect to my drum sound to increase the sonic complexity of this passage by mixing the sound of delayed synthesized drums with delayed acoustic drums.

4.4 Participation

Blackwell et al's *participation* quality consists of "supporting ongoing musical activity by making contributions that do not detract from but rather enhance the current musical direction" [6]. This quality is similar to Rowe's concept of "musical coherence" in that it describes a machine improviser's ability to produce sounds that relatively fit the context of a given musical state. *Participation* or "musical coherence" is achieved via *Heretic's* ability to contextualize low-level sounds from the human performer into a cohesive high-level organization structure during a real-time improvisation (Section 2.2.1) [62]. One example of *Heretic* participating in "ongoing musical activity" is in section 3, where it complements my *Granular Forming* by actuating a granular synthesizer that samples no-input mixer sounds. *Heretic's* leaving of sonic space for me to take a solo in sections 2 and 8 exemplifies its ability to "not detract from but rather enhance the current musical direction" [6]. Essentially, whenever *Heretic* engages in a *Unison* or *Complement* posture of interaction, its using the *Interpretive Listening* module to participate in and support musical gestures while not detracting from this musical gesture's organic manifestation.

4.5 Leadership

Live Algorithms obtain *leadership* by "[attempting] to change the musical direction, to invoke a new musical center" [6]. In this improvisation, *Heretic* shows leadership in sections 1, 9, and 11. In section 1, *Heretic* leads my playing from a *Drone Forming* to a *Melodic Forming* by autonomously actuating a *Melodic Forming* at the performance's onset. *Heretic* "[changes] the musical direction" in section 9 by subverting and intervening in my *Solo* posture of interaction with a series of erratic and obtrusive sound objects. This is also an example of "mutual subversion" [5]. *Heretic's* recalling of the *Drone Forming* from section 5 in Section 11 "[invokes] a new musical center" following my soft drum roll. This concept of *leadership* reinforces Lewis' concept of "emotional transduction" to take place [37]. When *Heretic* leads the improvisation into a new musical state, I must adjust my playing to maintain "musical coherence" despite an unexpected change in musical state [37, 62]. Since *Heretic* is still interpretively listening to me as I navigate my way through *Heretic's* periods of *leadership*, *Heretic* is then making contextual decisions or engaging in "emotional transduction" based on my reactions to its playing.

4.6 A New Live Algorithm

The above analysis and evaluation demonstrates that *Heretic* is a true Live Algorithm according to Blackwell et al's criteria, and adheres to the "aims of the designer," which are inspired by the previous work of Blackwell et al, Bailey, Rowe, and Lewis [5, 6, 37, 52, 62]. *Heretic interpretively listens* to a human performer, contextualizing low-level musical materials into a higher-level framework that enables a clear formal structure to emerge its interactions with a human performer. *Heretic's Contextual Decision Making* module enables *Heretic* to *autonomously participate* and *lead* in *novel* ways, which enables "mutual subversion" and "musical coherence" to take place during an improvisation [5,6,63]. These *Interpretive Listening* and *Contextual Decision Making* modules sonically manifest themselves in *Heretic's Musical Synthesis* module via its flexible approach to a *novel* musical output, thus enabling "emotional transduction" [37]. By objectively and aesthetically evaluating *Heretic's* musical output and interactions with a human performer, I can form conclusions regarding my work with *Heretic*, and identify new directions in Live Algorithms research and human-machine free improvisation.

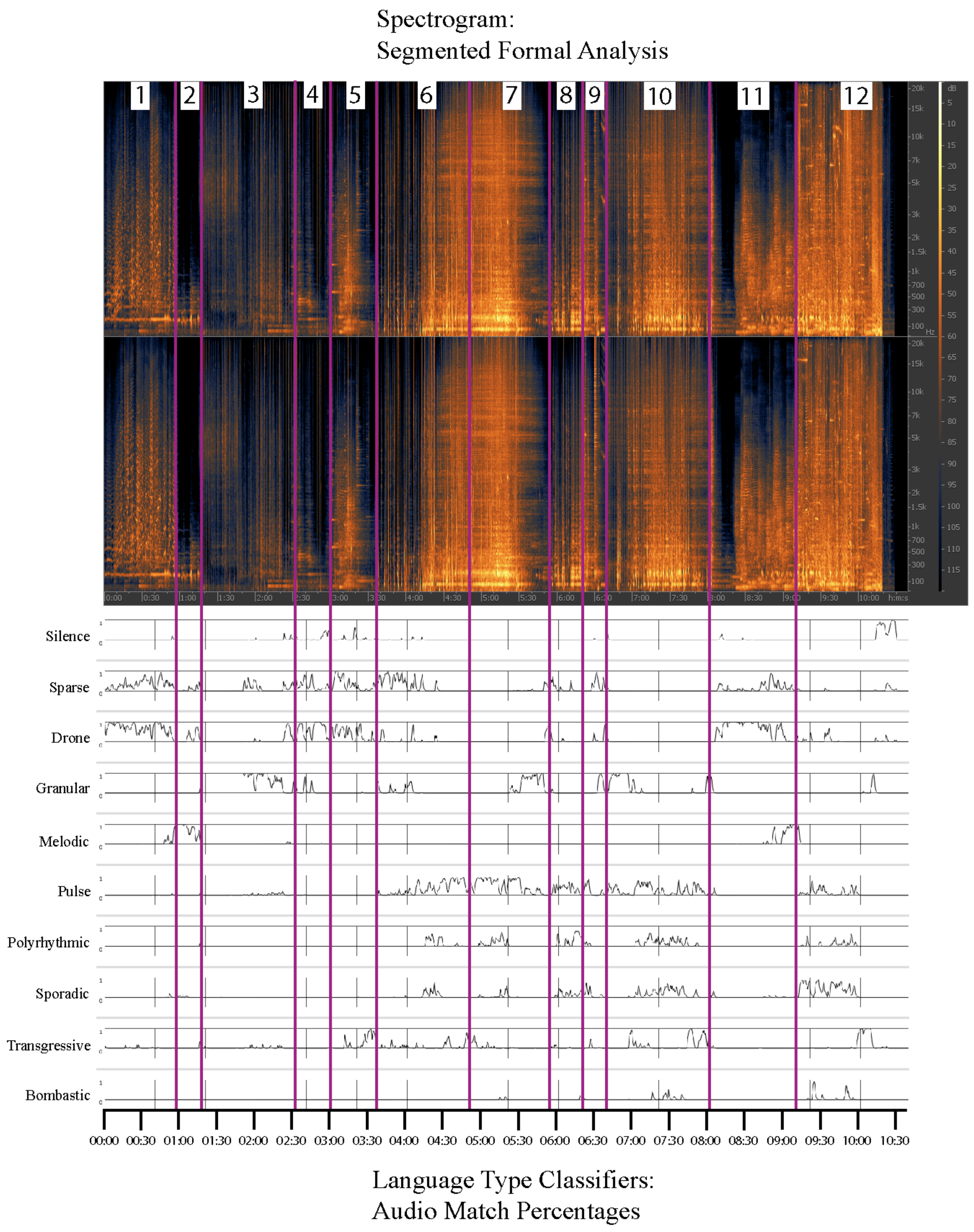


Figure 4.1: This figure's top portion is a spectrogram of an improvisation by *Heretic* and myself. This improvisation's formal sections are labelled as segments as discussed in Section 4.1. The bottom portion is the recorded output of the neural-network classifiers during the improvisation. Recording: https://soundcloud.com/hunter-brown-music/heretic-full-1

5 Conclusions and Future Work

The motivation in developing *Heretic* was to create a Live Algorithm that employs my own improvisational methodology as a conceptual and computational framework for human-machine free improvisation. This motivation particularly centers around my interests in Derek Bailey's "mutual subversion," Robert Rowe's "musical coherence," and George Lewis' "emotional transduction" [5, 37, 62]. Once establishing a motivation for creating a Live Algorithm, the study of Cecil Taylor's, Ornette Coleman's, Joe Morris', and Anthony Braxton's improvisational methodologies inspired me to formalize my improvisational methodology before encoding it as a computational framework within *Heretic* [9, 44, 73]. From this study, my improvisational methodology borrows and re-imagines concepts from

Morris' *Properties of Free Music*, and Braxton's *Language Music* due to their alignment with my particular aesthehic sensibilities. However, considering *Heretic's* role as a laptop improviser and my background in electroacoustic music composition, I also borrowed the following concepts when formalizing my improvisational methodology: Smalley's *structural functions*, Road's *theory of musical timescales*, and Wishart's *sequences and fields* [59,70,79].

Upon developing a conceptual basis for creating *Heretic*, I begin researching previous Live Algorithms as a means for learning how to implement these theoretical concepts into a computer music system via the SuperCollider and Wekinator software platforms. The work of Blackwell, Young, Bown, Lewis, and RepMus demonstrates a wide variety of approaches for designing a Live Algorithm [6,35,37]. In particular, I re-imagined Blackwell et al's proposed *PQF* architecture as a triple-layer architecture consisting of modules for *Interpretive Listening*, *Contextual Decision Making*, and *Musical Synthesis* [6]. These three modules work together to form a computational representation of my improvisational methodology while maintaining the motivational concepts of "mutual subversion," "musical coherence," and "emotional transduction" in *Heretic's* musical output [5, 37, 62]. Thus, *Heretic* "listens, reflects, selects, imagines and articulates its musical thoughts as sound in a continuous process" in a manner that serves my conceptual and aesthetic interests, while also challenging me as its improvisational partner [6]. The sounds that *Heretic* uses to "[articulate] its musical thoughts" are inspired by the music of Sam Pluta/Peter Evans, Frank Rosaly, and Toshimaru Nakamura/Kim Cascone [6,55,61]. *Heretic's* sonic voice is inspired by these improvisers' music because their work fits with my sonic interests in sonic cohesion via live processing, unruly noise oriented feedback, drones, micro-sonic glitch, varying rhythmic density, indeterminacy, and "failure" [11].

Once *Heretic* obtained the ability to express its musical voice autonomously, I began to think deeply about my improvisational interactions with *Heretic* and how these interactions fit with my intentions in developing this system. Chapter 4 details a formal analysis of an improvisation with *Heretic* using Blackwell et al's evaluative framework of "autonomy, novelty, participation and leadership" [6]. This analysis yielded the conclusion that *Heretic's* output adheres to Blackwell's et al defining criteria of a Live Algorithm and my musical and aesthetic intentions as defined in Chapters 1 and 2. However, what are the conclusions to be drawn from my personal experience as a human musician performing with *Heretic*? Was I able to predict what *Heretic* was going to do next? Was *Heretic* challenging me to spontaneously create novel musical materials and sequences? Is *Heretic* musically gratifying to perform with?

I found that *Heretic's* ability to achieve "mutual subversion" challenged me greatly as an improviser. I never knew what to expect from moment to moment, therefore I found myself playing musical material on the drum-set that I had never played before. I believe this is because that by being pushed out of my musical comfort zone, I was forced to spontaneously adapt to the given musical context with novel musical materials. This is a quality I look for in other human musicians that I improvise with. For me, the whole purpose and beauty of free improvisation is to enter mental states that enable truly novel and idiosyncratic musical interactions to take place. By way of these idiosyncratic musical interactions, the resultant music is likely to be music that has never been played before. As a musician, creating previously unheard music is at the crux of my overall goals as an artist. *Heretic's* ability to enable this kind of free improvisation to take place is a testament to its effectiveness as a Live Algorithm.

However, would other improvisers feel the same way playing with *Heretic*? While *Heretic* is primarily designed to serve as a musical device for my own improvisation practice, future work with *Heretic* will focus on re-imagining *Heretic's* design to work with other performers. Ideally, *Heretic* could function as a way for other improvisers to interact with my improvisational methodology within the world of interactive electroaouctic music. This would involve redesigning the *Interpretive Listening* module to easily adapt to any instrument without retraining its neural networks. Currently, one could retrain the language type classifiers to hear any musical language, however I think it would be worthwhile to find methods for enabling *Heretic* to interact with any instrument and any improvisational methodology without a priori training on the particular musical language of the person it is performing with. Aside from extending *Heretic's Interpretive Listening* module to robustly interact with any instrument without prior training, I think it would be interesting to extend *Heretic's* functionality to include interactions with groups of improvisers.

Most of the Live Algorithm's discussed in Chapter 2 have mainly been used in duo musical contexts with a solo human performer. Further iterations of *Heretic* will include functionalities that enable *Heretic* to participate within varying instrumentations and combinations of improvisational languages. In particular, it will be important for *Heretic* to alter its *Interpretive Listening* and *Contextual Decision Making* modules depending on the size of the ensemble and the instrumentation of the ensemble. For instance, if *Heretic* knows that it is performing in a large ensemble, it will have to *intrepretively* listen to each individual member of the ensemble and the collective output of the ensemble to better formulate its *Contextual Decision Making* process. The future possibility for *Heretic* to function in group improvisation will present more combinations of complex interaction with multiple human performers simultaneously, thus resulting in more idiosyncratic interactions amongst the other human performers via "mutual subversion" and "emotional transduction" [5,37]. Similar to how *Heretic* is currently trained on my musical language, I am especially interested in the possibility for *Heretic* to learn and interact with a collective group's musical language. Playing in a duo setting versus a group setting is a much different experience and practice, and by further developing *Heretic* to function in a greater variety of musical settings, I believe there is potential for interesting, challenging, and aesthetically gratifying improvised music making to take place.

The process of creating a Live Algorithm modelled after my own improvisational methodology has forced me to deeply examine my own musical practice. This journey has also enabled me to form a deeper understanding of improvisation at large, and how computers can act autonomously in musical settings. In his text *Towards an Ethic of Improvisation*, Cornelius Cardew wrote, "This kind of thing happens in improvisation. Two things running concurrently in haphazard fashion suddenly synchronize autonomously and sling you forcibly into a new phase" [10]. *Heretic* and I are "two things running concurrently" during a performance, and within our long term collaboration [10]. Both of our musical languages are in perpetual concurrent growth that enables us to collectively enter "a new phase" of music making during each performance [10]. Through our collective musical language, *Heretic* and I are "[synchronizing] autonomously" as improvisation partners. This synchronization of our musical languages enables *Heretic* and I to spontaneously create novel and idiosyncratic music that pushes the boundaries associated with improvised music and expands our abilities as individual improvisers.

Bibliography

[1] M. Adkins. *Extending the Instrumental Sound World Using Electronics*, pages 258–273. Cambridge Companions to Music. Cambridge University Press, 2 edition, 2017. DOI 10.1017/9781316459874.016. Citation on page 53.

[2] B. Alterhaug. Improvisation on a triple theme: Creativity, jazz improvisation and communication. *Studia Musicologica Norvegica*, 30(3):97–117, 2004. Citation on pages 2 and 14.

[3] C. Ames. The markov process as a compositional model: A survey and tutorial. *Leonardo*, pages 175–187, 1989. Citation on page 38.

[4] G. Assayag and S. Dubnov. Using factor oracles for machine improvisation. *Soft Comput.*, 8(9):604–610, Sept. 2004. DOI 10.1007/s00500-004-0385-4. Citation on pages 12 and 39.

[5] D. Bailey. *Improvisation: its nature and practice in music*. British Library National Sound Archive, 1992. Citation on pages 2, 3, 6, 14, 19, 24, 28, 39, 48, 62, 63, 66, 67, 69, 70, and 72.

[6] T. Blackwell, O. Bown, and M. Young. Live algorithms: towards autonomous computer improvisers. In *Computers and creativity*, pages 147–174. Springer, 2012. Citation on pages 5, 7, 8, 59, 60, 63, 64, 65, 66, 67, 70, and 71.

[7] O. Bown. Experiments in modular design for the creative composition of live algorithms. *Computer Music Journal*, 35(3):73–85, 2011. Citation on pages 7 and 8.

[8] O. Bown and S. Lexer. Continuous-time recurrent neural networks for generative and interactive musical performance. In *Applications of Evolutionary Computing*, pages 652–663, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. Citation on pages 5 and 34.

[9] A. Braxton. *Tri-Axium Writings*. Synthesis Music, 1985. Citation on pages 2, 6, 13, 14, 48, 62, and 69.

[10] C. Cardew. Towards an ethic of improvisation. *Treatise handbook*, pages 17–20, 1971. Citation on page 73.

[11] K. Cascone. The aesthetics of failure: "post-digital" tendencies in contemporary computer music. *Computer Music Journal*, 24(4):12–18, 2000. Online at http://www.jstor.org/stable/3681551. Citation on pages 24 and 70.

[12] M. Casey. Soundspotting: A new kind of process? In *The Oxford Handbook of Computer Music*. Oxford University Press, 2009. Citation on page 50.

[13] N. Collins. Scmir: A supercollider music information retrieval library. In *ICMC*, 2011. Citation on page 29.

[14] N. Collins. Noise music information retrieval. *Noise in and as music*, pages 97–117, 2013. Citation on page 14.

[15] N. Collins. Slugens: set of supercollider plug-ins of non-standard sound synthesis experiments, 2018. Online at https://composerprogrammer.com/code.html. Citation on page 49.

[16] A. Cont, S. Dubnov, and G. Assayag. A framework for Anticipatory Machine Improvisation and Style Imitation. In *Anticipatory Behavior in Adaptive Learning Systems*

(ABiALS), Rome, Italy, 2006. ABIALS. Online at https://hal.inria.fr/hal-00839074. Citation on page 12.

[17] D. Cope. *The algorithmic composer*, volume 16. AR Editions, Inc., 2000. Citation on page 39.

[18] C. Cox and D. Warner. *Audio Culture, Revised Edition: Readings in Modern Music*. Bloomsbury Publishing, 2017. Online at https://books.google.com/books?id=rxcqDwAAQBAJ. Citation on page 23.

[19] C. Dobrian. Strategies for continuous pitch and amplitude tracking in realtime interactive improvisation software. In *Proceedings of the 2004 Sound and Music Computing conference (SMC04)*, volume 15, 2004. Citation on page 32.

[20] C. Dobrian. Realtime stochastic decision making for music composition and improvisation. *Pendragon Press*, 2012. Citation on page 39.

[21] C. Dodge and T. A. Jerse. *Computer Music: Synthesis, Composition and Performance*. Macmillan Library Reference, 2nd edition, 1997. Citation on page 39.

[22] C. Duxbury, M. Sandler, and M. Davies. A hybrid approach to musical note onset detection. In *Proc. Digital Audio Effects Conf.(DAFX,âĂŹ02)*, pages 33–38, 2002. Citation on page 30.

[23] D. Eck and J. Schmidhuber. Finding temporal structure in music: Blues improvisation with lstm recurrent networks. In *Neural Networks for Signal Processing, 2002. Proceedings of the 2002 12th IEEE Workshop on*, pages 747–756. IEEE, 2002. Citation on page 34.

[24] A. Eigenfeldt, O. Bown, A. R. Brown, and T. Gifford. Flexible generation of musical form: beyond mere generation. In *Proceedings of the seventh international conference on computational creativity*, pages 264–271, 2016. Citation on page 16.

[25] A. Eigenfeldt, A. Burnett, and P. Pasquier. Evaluating musical metacreation in a live performance context. In *Proceedings of the Third International Conference on Computational Creativity*, pages 140–144. Citeseer, 2012. Citation on page 60.

[26] R. Fiebrink, D. Trueman, and P. R. Cook. A meta-instrument for interactive, on-the-fly machine learning. In *NIME*, pages 280–285, 2009. Citation on pages 1, 27, 29, and 35.

[27] F. Fontana. Preserving the structure of the moog vcf in the digital domain. In *ICMC*, 2007. Citation on page 55.

[28] J. Harkins. A practical guide to patterns. *Help Documentation: SuperCollider*, 3(4), 2009. Citation on page 51.

[29] P. Herrera, A. Dehamel, and F. Gouyon. Automatic labeling of unpitched percussion sounds. In *Audio Engineering Society Convention 114*, Mar 2003. Online at http://www.aes.org/e-lib/browse.cfm?elib=12599. Citation on page 32.

[30] L. A. Hiller and L. M. Isaacson. *Experimental music: composition with an electronic computer*. McGraw-Hill, 1959. Citation on page 38.

[31] M. Hoffman and P. Cook. Real-time dissonancizers: Two dissonance-augmenting audio effects. In *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, 2008. Citation on page 32.

[32] A. Jordanous. Evaluating evaluation: Assessing progress in computational creativity. *Proceedings of the 2nd International Conference on Computational Creativity (ICCC-11)*, 2011. Citation on page 60.

[33] B. Lévy. Omax documentaion. *IRCAM*, 2012. Citation on pages 11, 12, and 53.

[34] B. Lévy. *Principes et architectures pour un systéme interactif et agnostique dédié á l'improvisation musicale*. PhD thesis, L 'Université Pierre Et Marie Curie, 2013.

Thése de doctorat dirigée par Assayag, Gérard Informatique Paris 6 2013. Citation on pages 2, 10, and 11.

[35] B. Lévy, G. Bloch, and G. Assayag. OMaxist Dialectics. In *New Interfaces for Musical Expression*, pages 137–140, Ann Arbor, United States, May 2012. Online at https://hal.archives-ouvertes.fr/hal-00706662. Citation on pages 5, 11, 12, 21, 53, and 70.

[36] G. E. Lewis. Improvised music after 1950: Afrological and eurological perspectives. *Black Music Research Journal*, 16(1):91–122, 1996. Online at http://www.jstor.org/stable/779379. Citation on page 9.

[37] G. E. Lewis. Too many notes: Computers, complexity and culture in voyager. *Leonardo Music Journal*, 10:33–39, 2000. DOI 10.1162/096112100570585. Citation on pages 5, 9, 10, 12, 21, 28, 53, 66, 67, 69, 70, and 72.

[38] G. E. Lewis. Live algorithms and the future of music. *CT Watch Quarterly*, 30(2):19–24, 2007. Citation on page 10.

[39] G. Lock. *Forces In Motion: The Music And Thoughts Of Anthony Braxton*. Da Capo Press, 1989. Citation on pages 6, 13, 15, 16, 18, 35, 48, and 62.

[40] B. Logan et al. Mel frequency cepstral coefficients for music modeling. In *ISMIR*, volume 270, pages 1–11, 2000. Citation on page 32.

[41] G. B. Mazzola and P. B. Cherlin. *Flow, Gesture, and Spaces in Free Jazz: Towards a Theory of Collaboration*. Springer Publishing Company, Incorporated, 2009. Citation on pages 2 and 14.

[42] J. McCartney. Supercollider documentation, 2010. Citation on page 49.

[43] R. Memisevic. An introduction to structured discriminative learning. Technical report, Technical report, University of Toronto, Toronto, Canada, 2006. Citation on page 35.

[44] J. Morris. *Perpetual Frontier: The Properties of Free Music*. Riti, 2012. Citation on pages 2, 6, 13, 19, 27, 48, 52, 57, 63, and 69.

[45] M. C. Mozer. Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multi-scale processing. *Connection Science*, 6(2-3):247–280, 1994. Citation on page 34.

[46] M. C. Mozer and T. Soukup. Connectionist music composition based on melodic and stylistic constraints. In *Advances in Neural Information Processing Systems*, pages 789–796, 1991. Citation on page 34.

[47] T. Nakamura. Biography: Toshimaru nakamura. *http://www.toshimarunakamura.com*, 2018. Citation on pages 24 and 56.

[48] T. Nakamura and K. Cascone. *Color Quanta*. Silent Records, 2016. Citation on page 24.

[49] A. Y. Ng and M. I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes, 2001. Citation on page 35.

[50] J. Nika, M. Chemillier, and G. Assayag. Improtek: Introducing scenarios into human-computer music improvisation. *Comput. Entertain.*, 14(2):4:1–4:27, Jan. 2017. DOI 10.1145/3022635. Citation on pages 10 and 53.

[51] F. Pachet. The continuator: Musical interaction with style. *Journal of New Music Research*, 32:333–341, 08 2010. Citation on pages 2 and 53.

[52] M. Pearce and G. Wiggins. Towards a framework for the evaluation of machine compositions. In *Proceedings of the AISBâĂŹ01 Symposium on Artificial Intelligence and Creativity in the Arts and Sciences*, pages 22–32, 2001. Citation on pages 60 and 66.

[53] A. Pease, D. Winterstein, and S. Colton. Evaluating machine creativity. In *Workshop on creative systems, 4th international conference on case based reasoning*, pages 129–137, 2001. Citation on page 60.

[54] G. Peeters, B. L. Giordano, P. Susini, N. Misdariis, and S. McAdams. The timbre toolbox: Extracting audio descriptors from musical signals. *The Journal of the Acoustical Society of America*, 130(5):2902–2916, 2011. Citation on page 32.

[55] S. Pluta. *Laptop Improvisation in an Multi-dimensional Space*. PhD thesis, Columbia University, New York, NY, USA, 2012. AAI3507720. Citation on pages 22, 57, and 70.

[56] S. Pluta and P. Evans. *Event Horizon*. Carrier Records, 2014. Online at https://sampluta.bandcamp.com/album/event-horizon. Citation on pages 21, 22, 54, and 55.

[57] M. Puckette. *The theory and technique of electronic music*. World Scientific Publishing Company, 2007. Citation on page 55.

[58] R. Reed and R. J. MarksII. *Neural smithing: supervised learning in feedforward artificial neural networks*. Mit Press, 1999. Citation on pages 34 and 36.

[59] C. Roads. *Microsound*. Mit Press. MIT Press, 2004. Citation on pages 6, 13, 17, 63, and 70.

[60] F. Rosaly. Centering and displacement. *Utech Records*, 2012. Citation on pages 23 and 55.

[61] F. Rosaly. Biography: Frank rosaly. *https://www.frankrosaly.com*, 2018. Citation on pages 22, 55, and 70.

[62] R. Rowe. *Machine musicianship*. MIT press, 2004. Citation on pages 3, 8, 19, 48, 63, 65, 66, 69, and 70.

[63] R. Rowe. *Interactive Music Systems: Machine Listening and Composing*. MIT Press (MA), 2014. Citation on pages 2 and 67.

[64] A. M. Sarroff and M. A. Casey. Musical audio synthesis using autoencoding neural nets. In *ICMC*, 2014. Citation on page 34.

[65] E. D. Scheirer. Tempo and beat analysis of acoustic musical signals. *The Journal of the Acoustical Society of America*, 103(1):588–601, 1998. Citation on page 31.

[66] D. Schwarz. A system for data-driven concatenative sound synthesis. In *Digital Audio Effects (DAFx)*, pages 97–102, 2000. Citation on page 50.

[67] S. Senturk. *Computational modeling of improvisation in Turkish folk music using variable-length Markov models*. PhD thesis, Georgia Institute of Technology, 2011. Citation on page 39.

[68] J. Sietsma and R. J. Dow. Creating artificial neural networks that generalize. *Neural networks*, 4(1):67–79, 1991. Citation on pages 34 and 36.

[69] D. Smalley. Spectro-morphology and structuring processes. In *The language of electroacoustic music*, pages 61–93. Springer, 1986. Citation on page 17.

[70] D. Smalley. Spectromorphology: Explaining sound-shapes. *Org. Sound*, 2(2):107–126, Aug. 1997. DOI 10.1017/S1355771897009059. Citation on pages 6, 14, 15, 17, 27, 28, 44, 45, and 70.

[71] D. Stathakis. How many hidden layers and nodes? *International Journal of Remote Sensing*, 30(8):2133–2147, 2009. DOI 10.1080/01431160802549278. Citation on page 36.

[72] T. Stilson and J. O. Smith. Analyzing the moog vcf with considerations for digital implementation. In *Proceedings of the 1996 International Computer Music Conference, Hong Kong, Computer Music Association*, 1996. Citation on page 55.

[73] C. Taylor, E. G. Stevens, J. Lyons, K. McIntyre, H. Grimes, A. Silva, and A. Cyrille. *Unit structures*. Blue Note, 1987. Citation on pages 2, 14, and 69.

[74] C. Testa. Echo echo mirror house music. *Sound American*, 16, 2016. Online at http://soundamerican.org/sa_archive/sa16/sa16-echo-echo-mirror-house-music.html. Citation on page 57.

[75] P. M. Todd. A connectionist approach to algorithmic composition. *Computer Music Journal*, 13(4):27–43, 1989. Citation on page 34.

[76] I. Ulusoy and C. M. Bishop. Comparison of generative and discriminative techniques for object detection and classification. In *Toward Category-Level Object Recognition*, pages 173–195. Springer, 2006. Citation on page 35.

[77] S. Wilson, N. Collins, N. Collins, D. Cottle, and J. McCartney. *The SuperCollider Book*. Mit Press. MIT Press, 2011. Citation on pages 9, 27, 37, and 49.

[78] S. Wilson, D. Cottle, and N. Collins. *The SuperCollider Book*. MIT Press, 2011. Citation on pages 1 and 47.

[79] T. Wishart. *Audible design: a plain and easy introduction to practical sound composition*. Orpheus the Pantomime, 1994. Citation on pages 17 and 70.

[80] E. Wold, T. Blum, D. Keislar, and J. Wheaten. Content-based classification, search, and retrieval of audio. *IEEE multimedia*, 3(3):27–36, 1996. Citation on page 32.

[81] N. Wooley. Anthony braxton's language music. *Sound American*, 16, 2016. Online at http://soundamerican.org/sa_archive/sa16/sa16-language-music.html. Citation on pages 15 and 35.

[82] I. Xenakis. *Formalized music: thought and mathematics in composition*. Pendragon Press, 1992. Citation on pages ix, 38, 51, and 53.

[83] M. Young. Nn music: Improvising with a 'living' computer. In R. Kronland-Martinet, S. Ystad, and K. Jensen, editors, *Computer Music Modeling and Retrieval. Sense of Sounds*, pages 337–350, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. Citation on pages 2, 5, 21, and 34.

[84] D. Zicarelli. M and jam factory. *Computer Music Journal*, 11(4):13–29, 1987. Citation on page 39.

[85] Z. Zukowski and C. Carr. Generating black metal and math rock: Beyond bach, beethoven, and beatles. In *31st Conference on Neural Information Processing Systems (NIPS 2017)*, 2017. Citation on page 34.