A linguistic model of minimalist syntax composes *Tebe Poem*

Sean P. Anderson

The University of Michigan

Mentor: Somangshu Mukherji

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Bachelor of Sciences with Honors in Cognitive Science: Computation & Cognition from the University of Michigan
2020

Acknowledgements

There are always more people to thank than there is time and space, so I will do my best here. Thank you foremost to Somangshu Mukherji, for expanding cognitive science to places I feared it would never go, and for providing an avenue for my childhood dream of working to understand creativity. You have significantly shaped how I think about the world around me, as well as my career. Thanks also to Sam Epstein, who inspired me greatly with his humility and joy in encouraging the project.

There are many others who's guidance and mentorship presents themselves in threads of this project, as well as significantly shaped the thinker and person I am today:

Thank you, Taraz Lee, for being one of the best mentors I could ever ask for. Thanks for teaching me nearly everything I know about science, listening to all of my ideas, including the bad ones, always taking the time to help me with whatever it is I'm doing, and bringing joy to the work. Thank you, Colleen Seifert, for being a ceaseless advocate, listening to the ideas I didn't tell Taraz, and opening all the doors, including those I didn't even know existed. You both have shaped my career and abilities incredibly. I am forever in debt!

Thank you to the Weinberg Institute; your funding made so much of my learning (and travel) possible, as well as your never-ending encouragement and flexibility. I never thought I'd be so proud to be a part of a major! Thank you also to Rick Lewis, who put the seemingly opaque and globular mess of cognitive science into clearer perspective. Thank you CogSci Community, and Anne, Grayson, Nick, Ross, and Ty, who compassionately shared my passion for the mind; I'll forever cherish the long hours of debate we spent (and spend!).

Thank you, Adam Unsworth and Bryan Kennedy, for supporting me in becoming a better musician than I thought possible, and for always believing in my ability. Thanks for showing me just how incredible and thoughtful teachers can be, and for always going the extra mile for me, every single time, no matter what. Thank you to the Horn studio, for being the most awesome and talented group of people I've ever been a part of, and for always raising the bar of kindness and encouragement.

Thank you, YoungEun Yook, for teaching me there's plenty more than a single way to think, see, and feel. Thank you for your endless support and personal encouragement. Thanks also to those at NELP, who continue to open my mind every time we meet.

Thanks to my Packard chefs: Chuck, John, Roseanne and Priya. You kept me fed, sane, and happy, and I am so grateful.

And, thank you, most of all, to my family, who did everything else: Mom, Dad, Neal, and Mohra.

Abstract

For centuries, thinkers of many disciplines have noted the similarities between music and language, particularly their communicative power and cultural significance. Recent work on the Identity Thesis for Music and Language (Katz & Pesetsky, 2011; Mukherji, 2014) provides fertile ground for formalizing and simulating computational, process-level models of musical composition, although (to the author's knowledge) no such simulations have been attempted within the Minimalist Program (Chomsky, 2014b). This work proposes a model of Western tonal composition congruent with current mathematical formalizations of Minimalist syntax in linguistics (Collins & Stabler, 2016) and demonstrates its ability to generate valid musical surfaces previously composed by humans (Dmitri Bortniansky's *Tebe Poem*). In the light of current artificial systems' difficulties in computational creativity (Herremans, et al., 2017) and statistical deep learning attempts of musical composition (e.g. Huang, et al., 2019), generative models such as the one proposed provide avenues for future research leading not only to more interpretable and musically compelling artificial composition systems, but also sounder theories of musical composition and language production in humans.

Keywords: music, syntax, generative linguistics, minimalism, computational modeling

Author's Note

This paper attempts to theorize a procedure broadly used by the mind, across multiple disciplines (generative linguistics and music theory), and argues its implications reach even broader, including engineering. I am barely a student in each of these disciplines, so I apologize in advance if an unwarranted gross simplification or misconception is made in this paper. I assume the reader has a basic familiarity with generative linguistics, particularly concepts introduced in the Minimalist Program. I also assume the reader has a basic understanding of essential concepts in music theory, such as chords, triads, intervals, cadences, and others. I understand this is not really a fair assumption to make, so I've done my best to quickly explain more elusive concepts in the text. I hope that, nevertheless, you find it understandable and easy to follow.

A linguistic model of minimalist syntax composes *Tebe Poem*

Introduction

"Whither music?" Leonard Bernstein—a world-famous conductor and composer—famously asked this question to begin his Norton lectures, titled "The Unanswered Question," in 1973 (Bernstein, 1976). His talks and its reception added fuel to a debate lasting centuries on the similarities and differences of human languages and music. Two attendants of the lectures, Ray Jackendoff and Fred Lerdahl, met in the audience and continued to work on ideas inspired by Bernstein's comparison of music to generative linguistics, producing the widely known *A Generative Theory of Tonal Music* (henceforth GTTM) (Lerdahl & Jackendoff, 1983). With much insightful analysis Lerdahl and Jackendoff concluded, rather definitively, that music is not language, despite sharing some aspects of hierarchical structure present in generative theories of language (Chomsky, 2014a).

Despite this strongly worded conclusion, interest in the idea continued through various applications of generative theory to music, particularly in the Western Tonal tradition. Allan Keiler (1978) criticized the issues with Bernstein's approach to generative linguistics and music, and proposed new theories based on work by the music theorist Heinrich Schenker (Keiler, 1977). Namely, Keiler proposes a binary-branching structure present in music, made up of harmonic objects called *Stufen* (pl.) (Mukherji, 2014: 328). A *Stufe* (singular) is a "scale-step," or an abstract entity that acts as a precursor to the actual chords written in a composition, although their treatment with respect to theoretical analysis in tonal music was not exactly consistent throughout Schenker's career (Keiler, 1983; Mukherji, 2014). Importantly, Keiler analyzed Schenker's theory of the *Ursatz*, or the " 'fundamental structure' of a tonal piece" (Schenker

1979:6, as quoted in Mukherji, 2014:115), as a phrase structure with binary branching constituents (Keiler, 1983).

Meanwhile, the Chomskian generative linguistics tradition produced at least two significant changes in their approach, namely X-Bar theory (Chomsky, 1970) and later, the Minimalist Program (Chomsky, 2014b). By reshaping generative theory into a pursuit of efficiency and elegance in a truly derivational (rather than representational) system, the Minimalist Program enabled significant progress on the questions of the relationship between language and music (Mukherji, 2014). Namely, theories in the Minimalist Program do not assume *a priori* particular phrase structures but demonstrate that previously studied structural forms arise organically out of the repeated application of simple rules (Chomsky, 2014b).

Drawing upon the Minimalist Program for linguistics, the Identity Thesis for Language and Music was formulated (Katz & Pesetsky, 2011; Mukherji, 2014). The Identity Thesis stipulates firstly that music theory and linguistic theory are the same, and secondly that this is evidence for music and language being products of the same computational system present in the human mind (Mukherji, 2014). Katz & Pesetsky (2011) apply this theoretical lens to argue earlier GTTM theory is in fact equivalent to Minimalist syntax in linguistics, while at the same time positing that music does not have a lexicon. Katz & Pesetsky additionally contrast their theory with Rohrmeier's (2007) work with Context Free Grammars (CFGs) and tonal composition, arguing music theory concepts like "tonic" and "dominant" appear to be semantic in origin despite Rohrmeier's use of them directly in the grammars. Mukherji (2014) discusses a similar issue in Keiler's work analyzing Schenker's *Ursatz* (Keiler, 1983).

It would not be unfair to consider all of these works, regardless of their stance, to be a part of a growing body of literature on the Identity Thesis. Three of these works propose formal models for tonal composition (Katz & Pesetsky, 2011; Rohrmeier, 2007; Mukherji, 2014) which interestingly have all been supplied with their unique analysis and derivation of Bortniansky's *Tebe Poem*, an 18th century choral piece. However, limited additional examples have been provided from the Minimalist models of additional derivations. Additionally, to the author's knowledge, no model of tonal composition in the Minimalist paradigm has been computationally simulated in free composition for the purposes of advancing Minimalist theory of syntax in music. Such a study would advance progress towards a successful "type 2: Common properties of pieces within an idiom" (Katz & Pesetsky, 2011) grammar. It would unequivocally demonstrate which pieces are able to be generated and which are not, based on the outputs of such a model, effectively assigning all musical pieces between two categories: deviant and valid. Any discrepancy between such assignments and the existence of human-composed pieces within an idiom would motivate improvements in the theory. Additionally, if such a model were parameterized accordingly, it could make similar implications for Katz & Pesetsky's "type 3" level of generative description (2011).

Models of musical composition that are successful in this regard would also have implications for computational creativity. As music, like language, is infinitely productive (that is, there are infinitely many pieces that can be composed, even within most idioms), not every piece has been composed yet. A model like the one described could write music that it predicts as valid (i.e. interpretable) within an idiom and be used as artificially created artwork. In fact, one issue plaguing recent systems of artificial composition in music is the inability to generate pieces

with “long-term structure” (Herremans, et al., 2017). Despite pursuing a different goal, modeling work in the Identity Thesis paradigm seems aptly poised to conquer this challenge, especially as issues of semantics are tackled in both the Minimalist Program and in music.

Here we aim to provide the foundations for future work in simulating models of tonal composition within an Identity Thesis paradigm, and hence, the Minimalist Program of linguistics. We take a mathematical formalization of Minimalist syntax (Collins & Stabler, 2016), which was stipulated entirely in the domain of human language and makes no appeal to music, and with minimal modifications formalize two models of Minimalist tonal composition. We then simulate compositions from these models by turning them into a probability distribution over musical pieces and sampling from this distribution via simulating derivations. We demonstrate that our models are capable of composing pieces in the Western Tonal idiom, particularly Dmitri Bortniansky’s *Tebe Poem*, and producing additional pieces on their own.

Methods

Tasks

Two computational models of musical composition were simulated in simple Western tonal composition tasks. Results from the simulations are separated into two tasks. The first task, Tebe-Composition, is to generate the chords of Bortniansky's *Tebe Poem*, measures 9-16, given a Lexical Array of the *Stufen* present in the piece (see below). *Tebe Poem* was selected since it requires handling of chords outside of the tonic's scale, including a diminished chord, and was analyzed by three prior works within Identity Thesis research discussed above (Rohrmeier, 2007; Katz & Pesetsky, 2011; Mukherji, 2014). The second task, Open-Composition, is to generate other surface chord progressions starting from the same Lexical Array used for Task 1. This second task aims to demonstrate musical competencies and shortfalls of the models that would not be visible in the original task. All derivations are evaluated informally by the author. Each simulation produced results for both tasks, as the models yield Open-Compositions in search of a solution to Tebe-Composition. Compositions for both tasks were hand-picked by the author for the Results section.

Both of the models are simplifications of a prior mathematical formalization of Minimalist syntax (Collins & Stabler, 2016) which was designed entirely for human language and makes no discussion of music. All models were written and executed using Python (see Appendix for code). Throughout the simulations, the models assume octave and enharmonic equivalence; namely, different spellings of physical notes are considered equivalent (e.g. A# = Bb) and we leave out notation of the octave. Adding treatment of octaves would be trivial; treatment of enharmonic equivalence requires application of more advanced music theory, and

participation in some debates that are out of this paper's scope. Additionally, the models compose in the key of C Major, without loss of generalization to other keys; each output of the model could be simply transposed to another key. The same lexicon was used for both models.

Models

Lexicon

Unlike the model proposed in Katz & Pesetsky (2011), Mukherji (2014) hypothesizes that tonal music does in fact have a lexicon, i.e. one that differs from the lexicon used in language. This Lexicon is based on the music theoretical work by Schenker (1973) and defines *Stufen* as the inputs to both models. A *Stufe* is an abstract entity behind the scale and chords of a particular key (Schenker, 1973: 133-153), and has many implications for theoretical discussion in music outside the scope of this paper. For our purposes, *Stufen* (plural of *Stufe*) are defined more simply as below:

> **Definition: *Stufe***
>
> A *Stufe* is a 4-tuple <c5, c3, root, type> where "c5" is a circle of fifths feature, "c3" is a circle of thirds feature (both defined below), and "root" is the note name of the *Stufe* it represents, which essentially is the name of the chord that this *Stufe* will realize when uttered in a surface. "type" is one of {"Major", "minor", "diminished"} and will be discussed in more detail below.

c3 and *c5* serve as features of the *Stufe*, and are uniquely defined by the *root* and *type* of the object. We will use brackets to denote "accessing" a feature value from a *Stufe* object. For example, if C = <+00, +03, "C", "Major">, C[*c5*] = +00 (see definition of Filter below).

Importantly, each *Stufe* is related to each other by their placement within the Circle of Fifths, which organizes the Lexicon (Mukherji, 2014):

[Figure 1]

Two *Stufen* adjacent to another on the Circle of Fifths have roots exactly one perfect fifth apart. For example, on a piano keyboard, the notes C and G are five notes apart within the C Major scale, counting C as one. Importantly, the G *Stufe* serves as the V, or "dominant," functioning chord in C Major, while C Major serves as the I, or "tonic." A very common, if not most common, harmonic progression (i.e. sequence of chords) observed in tonal music is the sequence V-I (which, for theoretic purposes, is a type of "cadence"; for more information, see Aldwell & Schachter, 2011). Mukherji (2014) proposes a numerical feature for each *Stufe* based on its location in the Circle of Fifths (Figure 1). These features come in to play when the model decides how to order sequences of *Stufen* when composing.

> **Definition: *c5***
>
> *c5*, called a circle of fifths feature, is an integer value in the interval [-12, +12], which corresponds to a location on the Circle of Fifths. In this paper, "+00" will always refer to the position of the tonic.

The intuitive motivation behind the *c5* feature is that chords that have *Stufen* adjacent on the Circle of Fifths will be juxtaposed (i.e. via Merge, defined later) in the musical surface; in Western tonal music, adjacent *Stufen*, when placed, will be ordered in the negative direction (e.g. G = +01 to C = +00 and not the other way around). Since both triad chords and seventh chords

typically operate this way in tonal music, we hypothesize that both triads and seventh chords (e.g. C Major and C^7, which are both in *Tebe Poem*) are instantiations of the same *Stufe*, one of *type* "Major" (Mukherji, 2014). However, most chord progressions are not simple traversals of the Circle of Fifths. For instance, in the C Major scale, the F Major triad often functions as a IV chord leading to a V chord, earning the label "pre-dominant." A F *Stufe* could not be placed directly before a G *Stufe* if the direction and adjacency of *c5* features is required. Following Mukherji (2014), the models proposed here hypothesize that *Stufen* are also related by intervals of a third. Thirds-based relationships enable prolongations of the dominant like the one described (Mukherji, 2014). This progression could be made legal by a covert, or invisible, progression from F *Stufe* to a d *Stufe*. Looking at the Circle of Fifths, d has a *c5* value of +02, which could then lead to a G *Stufe*. This progression is hypothesized because F-d-G (functionally: IV-ii-V…) is in fact a chord progression observed often in Western tonal music, and there are reasons to believe overt composition of ii could have been dropped as musical style changed over time (see Mukherji, 2014: 359). For these reasons, the *c3* feature is defined:

Definition: *c3*

c3, or circle of thirds feature, is an integer value in the interval [-12, +12].

If a *Stufe* C is of *type* "Major," C[*c3*] = (*c5* + 3) mod 12, or the *c5* value the *Stufe* would have if its root was a minor third below its own.

If C is of *type* "minor," C[*c3*] = (*c5* + 3) mod 12 also, which is the *c5* value of its parallel major *Stufe*.

If C is of *type* "diminished," C[*c3*] = (*c5* + 8) mod 12, or the *c5* value *Stufe* would have if its root was a major third below its own.

The last part of this definition requires some explanation. Typically, diminished chords like F#º are followed by one of four chords, including G Major (the triad with root a half step up). With a minimal modification—the addition of D, the note a major third below the root F#—the diminished chord becomes a D^7, which would have a *c5* value of +02, adjacent to G's. The last of these four possible progressions takes place in *Tebe Poem.* The *c3* definition for diminished *Stufen* is a significant simplification of the complex harmonic (and voice-leading) nature of these chords; we leave defining a more accurate and elegant feature space for future work.

Model A

Model A is a much simplified version of a previous mathematical formalization of Minimalist Syntax which was formulated entirely within linguistics, and makes no appeal to musical phenomena nor the Identity Thesis (Collins & Stabler, 2016). In this section, we directly quote several definitions from the formalization, replacing the number of the definition with the name of the object defined (Collins & Stabler, 2016); for the reader's convenience, an asterisk (*) is added to each definition which was altered or introduced for our models. The model's input is a Lexical Array, which contains the *Stufen* required for the piece, and its output contains a binary-branching tree structure (a Syntactic Object) linking *Stufen,* and its corresponding chord sequence string read from the leaves of the tree.

***Definition: lexicon**

A lexicon is a finite set of *Stufen*.

***Definition: lexical item token**

A *lexical item token* is a pair <C, *k*> where C is a *Stufe* and *k* is an integer.

Definition: lexical array

A *lexical array* (LA) is a finite set of lexical item tokens. (p. 45)

Following the formalization in Collins & Stabler (2016), we define the objects necessary for the simple model nearly exactly the same way:

***Definition: syntactic object**

X is a *syntactic object* iff

(i) X is a lexical item token, or

(ii) X is an ordered pair of syntactic objects.

Definition: immediately contains relation

Let A and B be syntactic objects, then B *immediately contains* A iff $A \in B$.

Definition: stage

A *stage* is a pair S = <LA, W>, where LA is a lexical array and W is a set of syntactic objects. We call W the *workspace* of S. (p. 46)

Definition: root

For any syntactic object X and any stage S = <LA, W> with workspace W, if $X \in W$, X is a *root* in W. (p. 47)

Altering Syntactic Objects to ordered pairs makes it simpler to define Agree (see Model B) by dispensing with Triggers and using a simpler Labels relation (see Collins & Stabler, 2016: 63). We also do not need the more general "contains" relation for the model's ability to generate, since we only allow External Merge (see below). Objects are manipulated by two operations:

Definition: Select

Let S be a stage in a derivation S = <LA, W>.

If lexical token $A \in LA$, then $Select(A, S) = \langle LA - \{A\}, W \cup \{A\} \rangle$.

***Definition: Merge**

Given any two distinct syntactic objects A, B, Merge(A, B) = (A, B). (p. 47)

Importantly, the result of Merge is a Syntactic Object as we defined. External Merge in these models serves as a special case of Merge in Minimalist linguistic literature. Normally, Merge would operate on any two distinct Syntactic Objects, which would allow items contained ("contained" as Collins & Stabler (2016:46) define it) within other items to be merged together, resulting in "Internal Merge." It is theorized in Minimalist linguistics that the Internal Merge case is how movement transformations take place (Collins & Stabler, 2016). These models deal only with External Merge, and we leave the implications of Merge and all its cases for future work.

***Definition: derivation**

A *derivation* from lexicon L is a finite sequence of stages $\langle S_1, \ldots, S_n \rangle$ for $n \geq 1$, where each $S_i = \langle LA_i, W_i \rangle$, such that

(i) For all LI and k such that $\langle LI, k \rangle \in LA_1$, $LI \in L$,

(ii) $W_1 = \{\}$ (the empty set),

(iii) for all i, such that $1 \leq i \leq n$, either

(derive-by-Select) for some $A \in LA_i$, $\langle LA_{i+1}, W_{i+1} \rangle = Select(A, \langle LA_i, W_i \rangle)$, or

(derive-by-Merge) $LA_i = LA_{i+1}$ and the following conditions hold for some A, B:

(a) $A \in W_i$,

(b) W_i immediately contains B,

(c) Agree(A, B) = True, and

(d) $W_{i+1} = (W_i - \{A, B\}) \cup \{Merge(A, B)\}$. (p. 49)

Definition: derivable relation

A syntactic object A is *derivable from lexicon* L iff there is a derivation $<<LA_1, W_1>, \ldots, <LA_n, W_n>>$, where $LA_n = \{\}$ and $W_n = \{A\}$. (p. 50)

Our definition of derivation differs from that of Collins & Stabler (2016) in that only External Merge is allowed (via condition (b) of derive-by-merge) and that Merge only applies when two Syntactic Objects satisfy Agree (defined below). A derivation describes the sequence of operations it took to generate a musical surface. The derivable relation is formalized purely to illustrate the possibilities of studies on "type 2" grammars (Katz & Pesetsky, 2011) labeling some musical strings valid (i.e. derivable) and all others deviant. For instance, all musical strings generated by our models in this paper are derivable given our *Stufen* lexicon.

At this point the models begin to differ substantially from the formalization. Before we can determine what is derivable given a *Lexicon* or a specific *Lexical Array*, we need to define how the derivation would finish, and when Merge is applicable:

***Definition: Agree (Model A version)**

Given any two distinct syntactic objects A and B, Agree(A, B) = *True*.

Agree is formalized here to illustrate that Model B (discussed below) is a parameterization of Model A. Effectively, Model A does not use Agree, which results in External Merge's operation on any pair and ordering of Syntactic Objects. To determine when a derivation has lead to a viable musical surface, we first edit the Label operation present in the Collins & Stabler formalization (2016):

***Definition: Label**

Label is a syntactic function from syntactic objects to lexical items tokens, defined in the following way:

(i) For all lexical item tokens LI, Label(LI) = LI.

(ii) Let W be a derivable workspace. If (A, B) is contained in W, then Label((A, B)) = Label(B). (p. 65)

In the case of Syntactic Objects containing other Syntactic Objects, which contain others, and so on, Label becomes a recursive operation which returns the *Stufe* at the deepest right-branching leaf of the tree defined by SO. By formulating Label this way and omitting Triggers, we dispense with "checking" of features during the course of a derivation. This effectively makes a *Stufe* always encourage the same Merges, regardless of how deep it is in the tree, or how many times it was merged previously. In the interest of efficiency, this eliminates the need for storing and manipulating features of a Syntactic Object during the course of a derivation. More important is that Label enforces the right member of a Merge as the "head" (Collins & Stabler, 2016:65), and therefore the models build entirely left-branching tree structures. With a small change this could operation could be parameterized to enforce the left member to be the head.

Next, we introduce a Filter operation:

***Definition: LeftLeaf**

LeftLeaf is a syntactic function from syntactic objects to lexical items tokens, defined in the following way:

(i) For all lexical item tokens LI, LeftLeaf(LI) = LI.

(ii) For all syntactic objects SO = (A, B), LeftLeaf(SO) = LeftLeaf(A).

***Definition: Filter**

Let SO = (H, I) and I = (J, K) be syntactic objects. Filter(SO) = *True* iff the following conditions are met:

(i) Label(SO)[*c5*] = +00,

(ii) Label(J)[*c5*] = +01, and

(iii) LeftLeaf(SO)[*c5*] = +00.

Filter(SO) = *False* otherwise.

LeftLeaf is formalized solely for notational simplicity when defining Filter. The novel Filter operation is the only segment of Model A that has specifically musical origins, aside the ordered parameterization of Label. Intuitively, Filter checks if the root of the Syntactic Object is Schenker's *Ursatz*, i.e. a I-V-I progression at the hierarchical level closest to the root (Schenker, 1979; Mukherji, 2014). This effectively enforces the piece to begin with the tonic and have a full cadence at the end. Earlier work in the Identity Thesis espouses the importance of cadences to determining listener's interpretation and even formalizes their requirement in models of musical syntax (Lerdahl & Jackendoff, 1983; Katz & Pesetsky, 2011). The *Ursatz* arises out of Schenker's extensive analysis of works in the Western Tonal musical style, where it was often exposed as the deepest level of harmonic structure in his analyzed works (Mukherji, 2014). For linguistic purposes, particularly within Minimalist paradigms, the *Ursatz* Filter acts to summarize third factor principles which encourage musical surfaces to contain the *Ursatz* structural form. However, it is outside the scope of this paper to hypothesize whether this principle or an equivalent group of principles would serve the same role in language.

Finally, we will now describe the actual procedure that enables Model A to produce musical surfaces:

***Definition: Derive (Model A version)**

Procedure Derive(LA) takes as input a Lexical Array of *Stufen* and proceeds (informally) as follows:

1) Instantiates a Stage S = <LA, {}>.

2) Picks two random *Stufen* from LA, and Selects them, moving them into the Workspace. S is now <LA - {A,B}, {A, B}>.

3) Flips a coin to decide whether to attempt External Merge (#5) or Select (#4).

4) If Select, picks a random *Stufe* from LA in current stage and performs Select to move that *Stufe* into the Workspace. If LA is empty, does nothing. Returns to Step #3.

5) If External Merge, picks two random Syntactic Objects SO1 and SO2 immediately contained in the Workspace, and performs Merge(SO1, SO2) to turn them into a new Syntactic Object SO3 in the Workspace. If Filter(SO3) returns True, executes SpellOut(SO3). If the Workspace contains less than two Syntactic Objects, does nothing. Returns to Step #3.

6) If at Step #3 and both LA is empty and the Workspace contains less than two Syntactic Objects, halts the procedure. If Workspace contains exactly one Syntactic Object SO and Filter(SO) returns True, the derivation is considered a "success." Otherwise, the derivation is considered a "crash."

***Definition: SpellOut**

Procedure SpellOut(SO) takes as input a Syntactic Object SO and proceeds (informally) as follows:

1) Compute, via left-to-right tree traversal of SO, the matching surface string of SO, which consists of each leaf *Stufen's root* name and *type* concatenated.

2) Print the surface string (the model's equivalent of "pronouncing" the surface).

As our goal is to simply generate viable musical surfaces, we do not formally model assigning an interpretation to the Syntactic Object generated, unlike GTTM (Lerdahl & Jackendoff, 1983), nor sending information to the CI interface, as posited in Minimalist linguistics (Collins & Stabler, 2016). An example of SpellOut is as follows:

Given SO = {C Major, {{d minor, G Major}, C Major} }

SpellOut(SO) = "C Major - d minor - G Major - C Major"

The attentive reader will note that by flipping coins to decide when to Select and when to External Merge, as well as which objects to operate on, effectively turns the process model into a probability distribution over musical surfaces and Syntactic Object trees. This presents some worthwhile avenues for further study of similar models in both linguistics and music, particularly with respect to "type 2" goals (Katz & Pesetsky, 2011: 5). This will be touched on in the Discussion.

Model B

Model B is formalized exactly the same as Model A, with some small modifications. The first is that Agree takes *Stufen* features into account, the second is that the Derive procedure only executes External Merge on two objects when they Agree, and the third is a necessary change to Derive to account for the increased situations where neither Select nor External Merge is possible. The goal of Model B is to mathematically approximate the model proposed in Mukherji

(2014) along the lines of Collins & Stabler's (2016) formalization of Minimalist syntax. Formally:

***Definition: Agree (Model B version)**

Operation Agree(SO1, SO2) (Model B version) operates on Syntactic Objects SO1 and SO2. Agree(SO1, SO2) = True if and only if:

1) Labels(SO1)[*c5*] - Labels(SO2)[*c5*] = 1 or 0, OR

2) Labels(SO1)[*c3*] - Labels(SO2)[*c5*] = 1 or 0, OR

3) Labels(SO2)[*type*] = "minor", SO2 is a *Stufe*, and Labels(SO1)[*c5*] - Labels(SO2)[*c3*] = 1 or 0.

Agree(SO1, SO2) = False otherwise.

Agree formally defines what kinds of Merges are possible. Condition #1 is a simple progression around the Circle of Fifths, e.g. G Major leading to C Major. Here, the *c5* features and the way Mukherji (2014) organized the lexicon begins to shape the musical surfaces composed by the model. Condition #2 allows progressions that are not explicit in the Circle of Fifths, including the common IV-V-I progression (e.g. F Major - G Major - C Major). Making use of *c3* features is justified by the fact that *Stufen* are related in more ways than the Circle of Fifths (Mukherji, 2014). Particularly, the F Major triad can become d minor through a simple 5-6 motion, which is a common voice-leading technique observed in music (Mukherji, 2014). This 5-6 motion is represented here by giving F Major *Stufen* the *c5* feature it would have if the root was moved down a minor third (see "Lexicon"), which in this case is the *c5* of D. Agreeing via *c3* features can also be interpreted as a covert (i.e. invisible on the surface) progression through the Circle of Fifths from the left *Stufe* to the right *Stufe* (Mukherji, 2014). However, this is a speculation, so

we leave the presence of 5-6 motion as the main justification for *c3* features and their handling by Agree.

Condition #3 is actually sort of an exception clause to enable Merges to minor chords from their dominant-functioning chords (e.g. E Major to a minor). This is required since we defined minor *Stufen* to have the same *c5* feature as their relative major *Stufen*, resulting in their *c3* features being equal to the *c5* feature of their parallel major *Stufen*. If *Stufen* cannot supply their *c3* features on the right side of a Merge to Agree*,* then common progressions such as E Major - a minor would not be possible. That being said, Condition #3 implies that this progression would occur from a major *Stufe* covertly to the next major *Stufe* on the Circle of Fifths, which then is (overtly) realized as the relative minor *Stufe*. From a voice-leading standpoint, this does not entirely make sense, since many ingredients (i.e. a leading tone) are present that make a progression like E Major - a minor smooth. The need for this condition (in order to generate *Tebe Poem*) presents an inelegant flaw in the model, and future work will address this issue further. Furthermore, a different formulation of Agree could parameterize this model to work in a different tonal idiom. For instance, Condition #1 could be changed to be: "Labels(SO2)[*c5*] - Labels(SO1)[*c5*] = 1 or 0," and the other conditions redefined accordingly. In fact, this parameterization would switch Model B from working in the Western Tonal tradition to the Rock tradition, based on work by Mukherji (2014). As we are only working in Western tonal music in this paper, we leave exploration of parameterizations of this type of model for future work.

Lastly, we slightly rewrite Derive to ensure that Agree is used soundly in the derivations:

***Definition: Derive (Model B version)**

Procedure Derive(LA) (Model B version) is defined exactly as the Model A version, except step #5 and step #6 are replaced with:

5) If External Merge, locates all possible ordered-pairs of Syntactic Objects SO1 and SO2 immediately contained in the Workspace for which Agree(SO1, SO2) = True. Picks randomly from these ordered pairs and execute Merge(SO1, SO2) to turn them into a new Syntactic Object SO3 in the Workspace. If Filter(SO3) returns True, executes SpellOut(SO3). If the Workspace contains less than two Syntactic Objects, or no ordered pair in the Workspace satisfies Agree, does nothing. Returns to Step #3.

6) If at Step #3 and both LA is empty and the Workspace contains either less than two Syntactic Objects or no ordered pairs of Syntactic Objects that satisfy Agree, halts the procedure. If Workspace contains exactly one Syntactic Object SO and Filter(SO) returns True, the derivation is considered a "success." Otherwise, the derivation is considered a "crash."

Results

Model A

Task 1

As described in Methods, we provided Model A with the requisite Lexical Array for *Tebe Poem*, which is:

LA = {C, C, F, D, G, E, a, F#º, G, C}

We executed Derive(LA) for 200,000 iterations or until the model correctly printed the surface chord progression string of *Tebe Poem*, whichever occurred first. As expected, Model A successfully composed *Tebe Poem*'s chords, yielding the tree structure (Figure 2):

[Figure 2]

Importantly, the tree structure does not correspond at all with interpretations of *Tebe Poem* in prior work (Katz & Pesetsky, 2011; Rohmeier, 2007; Mukherji, 2014). Notably, F#º is merged to a minor, while three previous analyses within the Identity Thesis paradigm show F#º merged directly to G Major. There is a strong precedence for this in harmonic analyses within music theory, as it demonstrates how diminished chords are often used: the leading tone in the bass resolves one half step up to the bass (and root) of the following chord in the string, G. In fact, a automated harmonic analysis system based on the Context Free Grammars (CFG) proposed by Rohrmeier (2007) assigns F#º to the phrase with a minor before (i.e. farther from the root node of the tree) the phrase with G Major (Bas De Haas, et al., 2013). But Model A's derivation of *Tebe* differs from the structure of that phrase by assigning G Major as leading to E

Major. E Major is shown in all four of the previous analyses discussed as leading to A Major; they would not predict D Major and G Major as being members of a phrase prolonging E Major, which serves as a left-branching prolongation of F#º. Nor, in the spirit of music theory, is this passage intuitively heard this way.

Clearly, this interpretation and its implications about each chord's prominence is problematic from a musical standpoint. If D-structure is believed to share some resemblance of the hierarchical and non-adjacent ways *Stufen* are related in a surface piece, at first glance it appears that Merge cannot apply willy-nilly as it is here. That is, in order to satisfy basic conventions of Western Tonal harmony, the *Filter* proposed in Model A is not enough to filter out deviant interpretations. However, for the Minimalist paradigm, this does not present a problem. The model simply labeled *Tebe Poem* as a valid musical surface, within the Western Tonal idiom. Whether the model will determine other chord progressions to be deviant or valid (by virtue of what it generates) is a question for Task 2.

Task 2

During simulations of Model A in generating *Tebe Poem*, other derivations reaching *SpellOut* (printed to the screen) were recorded. A handful of them picked by the author are discussed here (Figure 3, 4):

[Figure 3]

Above is the simplest and shortest surface possibly generated by Model A. It is a direct replication of the *Ursatz*, and is expected as a result of *Filter*.

[Figure 4]

While this derivation (Figure 4) successfully met the conditions of the *Ursatz* Filter, there are clear issues with the surface generated. Ignoring the grouping of surface chords into phrases, the piece finishes with F#º - C. This would be moving the root by a tritone (a very dissonant, crunchy sounding interval), which presents an issue for voice-leading. Additionally, the diminished VII chord (which F#º is to C Major) nearly never serves the function of dominant, let alone in a cadence ending a piece. However, because of the *Ursatz* Filter, Model A selects G several chords earlier as the main dominant-serving chord to the ending tonic, leaving F - C - D - F#º - C as a harmonic prolongation of the tonic. From both a voice-leading standpoint and a harmonic standpoint, this is deviant. This collection of chords would usually appear in a pre-dominant section of a piece. Additionally, the first four chords are labeled as a prolongation of E Major (since E Major is the head of that phrase) but neither C Major nor G Major are contained in the E Major scale, so that does not make sense either. In Open-Composition, Model A produces both viable and deviant chord progressions, likely by allowing more *Stufe* merge-pairs to be generated than is reasonable, resulting in overgeneration.

Model B

Task 1

Again, we provided Model B with the Lexical Array required to compose *Tebe Poem*. We ran the *Derive* procedure 200,000 times or until surface chord string of *Tebe* was printed, whichever occurred first. As this model closely aims to replicate the model described in Mukherji (2014), we expect *Tebe* to be generated (Figure 5):

[Figure 5]

As expected, Model B terminated with the surface string matching *Tebe Poem*. This tree structure nearly exactly corresponds to Rohrmeier's (2007) derivation of the same piece using a CFG, with the only difference being our handling of the first two *Stufen*, C Major and C^7, as separate entities in the starting Lexical Array (which results in the leftmost C-C-F constituent being a tonic prolongation in C in Model B, rather than a subdominant prolongation in the key of C as in Rohrmeier's model). In fact, this derivation from Model B matches Rohrmeier's (2007) analysis of all chords before F#º being a prolongation of a minor, rather than C Major which starts and ends the piece (although another model proposed in Rohrmeier (2011) claims the opposite). It is this interpretation that differs between Model B's generation and Mukherji's (2014) analysis of *Tebe Poem*. This is further evidence that the constituent structure (e.g. F#º leading correctly to G Major, E Major leading to a minor, etc.) much more closely matches what is heard by the listener, or even generated by the composer than the structure proposed by Model A. However, the only goal is to successfully generate the *Tebe* surface; *Tebe* is possible and labeled valid by its parameterization of Agree and Filter.

Task 2

Similarly to our simulations with Model A, we recorded some of the compositions reaching *SpellOut* in simulations of Model B. A handful are discussed here (Figure 6, 7):

[Figure 6]

Here we see all three different clauses of Agree at work. F#º Merges as the model intended to G Major, via the implied covert progression to D major encapsulated in the *c3* feature. This *c3* feature and its corresponding clause (#2) in Agree also result in a merge from F Major (*c5* = -01) to D Major (effectively a IV-ii progression in the key of C Major), which was also an intention of the *c3* design. We see also that the constituent tree headed by D Major (*c5* = +02) Merges with E Major (*c5* = +04). This is again, made possible by D Major's *c3* feature which is equal to +05, or the *c5* feature assigned to B Major. Clause #3 enables this E Major (*c5* = +04) constituent to merge with a minor, which uses its *c3* = +03 feature (borrowing from its parallel major, A Major, *c5* = +03).

[Figure 7]

This derivation (Figure 7) demonstrates some merges that may not be within the Western Tonal idiom, and issues ripe for future work in Minimalist composition models. On the surface, it does not make sense for C Major to travel to a minor and back again, without other chords enabling such a modulation. However, this is not immediately grasped from the tree structure as

a minor is buried within a constituent headed by D Major (serving a pre-dominant role). Furthermore, F#º is merged with D Major, which is not expected in Western Tonal music (S. Mukherji, personal communication, July 30, 2020) illustrating a potential flaw in Agree's *c3* clause for diminished chords (#2). F#º's *c3* feature is, like F# Major's, +02. D Major's *c5* is +02. Agree is quite flexible in allowing *Stufen* of equivalent features to merge together (which is necessary for the C Major to C Major merge at the root of Mukherji's (2014) analysis of *Tebe Poem*). In this case, we could try to repair the issue by hoping merges via equivalent feature values are covert progressions and F#º would not be "pronounced." This would potentially solve another possible issue with this derivation: E Major currently prolongs D Major via its merge to F#º (which was also enabled by Agree clause #2), which could be unexpected in the Western Tonal tradition. Without F#º's Merge to D Major, this type of prolongation would no longer be predicted by the model, which instead could find another way to prolong D Major in this piece.

Discussion

We hypothesized that a model of Minimalist linguistic syntax with no musical mechanics would be capable of composing a pre-existing work in Western Tonal music. Our two models demonstrated that this is indeed possible for Bortniansky's *Tebe Poem* with minimal modifications to the original model. Additionally, we demonstrated that our models both compose original (with respect to the model itself, not to human composition) chord progressions as well as overgenerate compositions, with a restrictive *Ursatz* Filter and Agree formulation. Our proposed lexicon, made of *Stufen* with numerical harmonic feature values, provides a starting point for successful Minimalist models of both musical and linguistic syntax.

Both Model A and Model B produced chord progressions beyond that of *Tebe Poem* in the Open-Composition task. We believe this is an advantage for our models, as successful Minimalist models should be able to compose valid pieces that have not been written before by humans. While Model B appears to be capturing some aspects of harmonic activity in Western Tonal music, there are some issues that present areas for improvement. For instance, in Model B's first Task 2 derivation (Figure 6), the author has the sensation that B Major needs to be overt (or with some clever voice-leading, introducing some tones of figuration) for the progression to E Major to be justified. Additionally the resolution of G Major is delayed for at least three chords before it deceptively cadences into a minor, and this takes a toll on the ear. Katz & Pesetsky (2011) propose a Tonal Harmonic Component (THC) which encourages Internal Merge to create cadences within constituent phrases. If Model B were to be modified to produce more interpretable musical surfaces, we would speculate that an Internal Merge (movement) of the a minor Stufe to the constituent immediately headed by G Major would solve the issue of delayed

cadence in Model B's piece. Perhaps this would be the movement encouraged by a modified version of Tonal Harmonic Component for this model.

The models' derivations (Model B in particular) were heavily driven by the nature of the Lexicon we defined. The *Stufen* lexicon bears theoretical assumptions about how precursors to musical harmonic activity might interact which were motivated by music theory (Mukherji, 2014). These assumptions draw upon the logic of a particular school of thought, specifically Schenkerian analysis, yet there are many other approaches to tonal analysis which would have differing implications about a lexicon in a Minimalist grammar (e.g. see Cohn cycles and Riemannian theory, Mukherji, 2014: 177-181).

While this study provided advances towards a successful "type 2" description of grammar (Katz & Pesetsky, 2011), Task 1 was rather limited in that it only tested one example. Future work should aim to test models of composition on more examples (as accomplished in Rohrmeier (2011) and others) or potentially a corpus of thousands of examples. However, the creation of such a corpus warrants a sizable effort, since decisions about which paradigm of harmonic analysis is desired as well as which pieces to group into a single idiom must be made. The difficulty of assembling such a corpus should not discourage simulation as a method of studying Identity Thesis theories. By formulating the models with a stochastic process, we effectively created a probability distribution over linguistic and musical surfaces. Models fashioned out of formalizations in generative linguistics and engaged with stochastic processes could easily be used if a probability greater than zero is assigned to a given surface, then that surface is predicted by the model to be a member of the parameterized idiom. By collecting a

corpus and representing the distribution of true musical surfaces, it could be easier to see where current theories are falling short and what modifications need to be made.

In tandem, Task 2 was limited in that we only gave the models the Lexical Array hypothesized for *Tebe Poem*. Future study should more thoroughly test the models' capabilities of open-composition by providing many different Lexical Arrays, possibly in a systematic manner. However, the models' ability to discover multiple pieces that can be composed with the same starting *Stufen* is certainly advantageous as it demonstrates that we have yet to understand fully the limits of musical composition models in the Minimalist paradigm.

Therefore, this study serves only as a proof of concept that Minimalist tonal composition is possible, and leaves a more thorough exploration and confirmation of the viability of Minimalist Identity Thesis theories of musical composition and language. For instance, any notion of phonological or semantic interpretability (i.e. "Full Interpretation," see Chomksy, 2014:24) is summarized in our use of the *Ursatz* Filter; while there is extensive work in Schenkerian music theory to justify its use (Mukherji, 2014), whether our models' *Ursatz* Filter corresponds at all to the hypothesized conceptual and phonetic interpretability of Western Tonal music remains a topic for further research. As we only used a simplified form of a single mathematical formalization, it is also likely that there are operations or qualities generally accepted within Minimalist linguistics that are not included or addressed here. Alterations to our models based on other work may significantly change the models' ability to perform. Future work should keep pace with current developments in the Minimalist Program to ensure the congruency of the model with linguistic syntax as well as emerging results in music theory.

This study has several implications for future research on the Identity Thesis as well as engineering systems for computational creativity. Our tasks, Tebe-Composition and Open-Composition, are rather naive ways to test the full capabilities and issues of the model as artificial composition systems. Better tasks, such as those that involve expert opinion (Wiggins, et al., 2009), need to be developed for effective development of successful systems. Meanwhile, Minimalist models of tonal composition seem well poised to conquer issues of long-term structure and are much more easily interpretable than deep learning systems (Herremans, et al., 2017). The challenge of writing pieces with compelling long-term structure may require a model of tonal composition that functions the way the mind is theorized to compose. In this way work in the Identity Thesis bridges three disciplines: engineering, generative linguistics, and music theory.

References

Aldwell, E., Schachter, C., & Cadwallader, A. (2011). Harmony and Voice Leading (4th ed.). *Cengage Learning*.

Bernstein, L. (1976). The unanswered question: Six talks at Harvard (Vol. 33). Harvard University Press.

Bortniansky, D. (1751-1825). *Tebe Poem.*

Chomsky, N. (2014a). Aspects of the Theory of Syntax (Vol. 11). MIT press.

Chomsky, N. (2014b). The minimalist program. MIT press.

Chomsky, N., Jacobs, R. A., & Rosenbaum, P. S. (1970). Remarks on nominalization. *1970*, *184*, 221.

Collins, C., & Stabler, E. (2016). A formalization of minimalist syntax. *Syntax*, *19*(1), 43-78.

De Haas, W. B., Magalhães, J. P., Wiering, F., & C. Veltkamp, R. (2013). Automatic functional harmonic analysis. *Computer Music Journal*, *37*(4), 37-53.

Herremans, D., Chuan, C. H., & Chew, E. (2017). A functional taxonomy of music generation systems. *ACM Computing Surveys (CSUR)*, *50*(5), 1-30.

Huang, C. Z. A., Cooijmans, T., Roberts, A., Courville, A., & Eck, D. (2019). Counterpoint by convolution. *arXiv preprint arXiv:1903.07227*.

Keiler, A. (1977). The syntax of prolongation I. *Theory Only*, *3*(5), 3-27.

Keiler, A. (1978). Bernstein's "The unanswered question" and the problem of Musical competence. *The Musical Quarterly*, *64*(2), 195-222.

Keiler, A. (1983). On some properties of Schenker's pitch derivations. *Music perception*, *1*(2), 200-228.

Mukherji, S. (2014). Generative Musical Grammar-A Minimalist Approach. *PhD Diss. Princeton University*.

Pesetsky, D., & Katz, J. (2011). The Identity Thesis for Language and Music. *Unpublished draft*.

Rohrmeier, M. (2007). A generative grammar approach to diatonic harmonic structure. In H. Spyridis, A. Georgaki, G. Kouroupetroglou, & C. Anagnostopoulou (Eds.), *Proceedings of the 4th Sound and Music Computing Conference* (pp. 97-100).

Rohrmeier, M. (2011). Towards a generative syntax of tonal harmony. *Journal of Mathematics and Music*, *5*(1), 35-53.

Schenker, H. (1973). Harmony, edited and annotated by O. Jonas, translated by EM Borgese.

Schenker, H. (1979). Free Composition:(der Freie Satz): Heinrich Schenker; Translated and Edited by Ernst Oster. Longman.

Wiggins, G. A., Pearce, M. T., & Müllensiefen, D. (2009). Computational modeling of music cognition and musical creativity (pp. 383-420). na.

Lerdahl, F., & Jackendoff, R. S. (1983). *A generative theory of tonal music*. MIT press.

Figures

Example 1.2-28. Major/minor triadic relationships represented as "Circle of Fifths" features

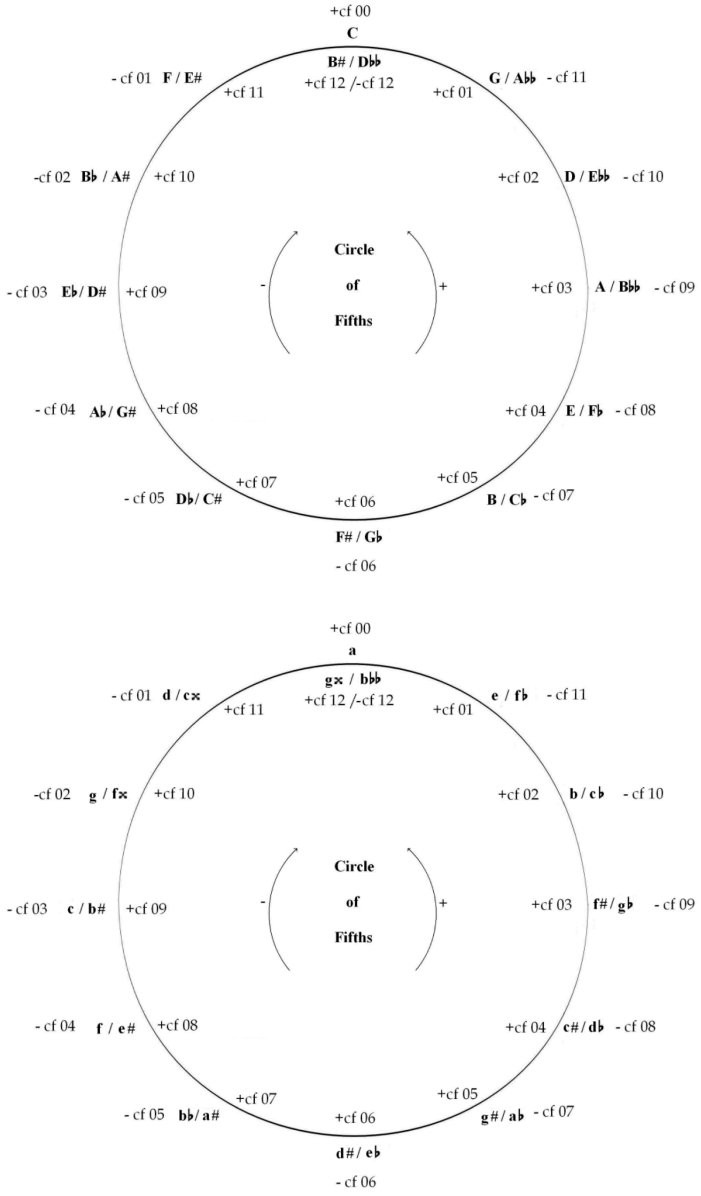

Figure 1. Circle of Fifths with *c5* features ("cf"), reprinted with permission from Mukherji (2014: 338). Each *Stufe* is related to others by intervals of a perfect fifth. Top: Major *Stufen*, Bottom: minor *Stufen*. Diminished *Stufen* and *c3* values not pictured.

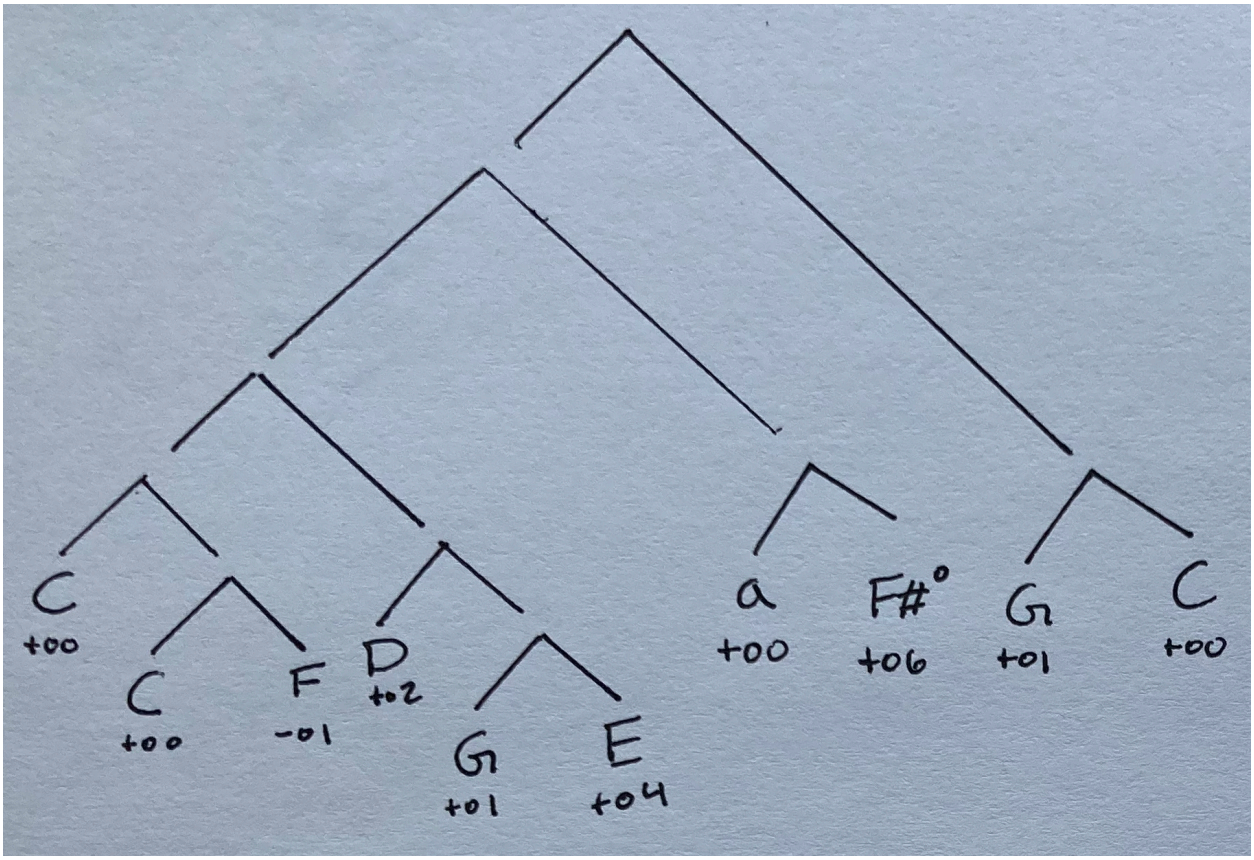

Figure 2. Model A, Task 1 (Tebe-Composition) derivation. Letters indicate *Stufen* root names; lowercase indicates minor, º indicates diminished, and capital indicates major types. Numbers indicate *c5* value. Model A successfully generates surface chord ordering of *Tebe Poem;* however, it posits a confusing harmonic structure.

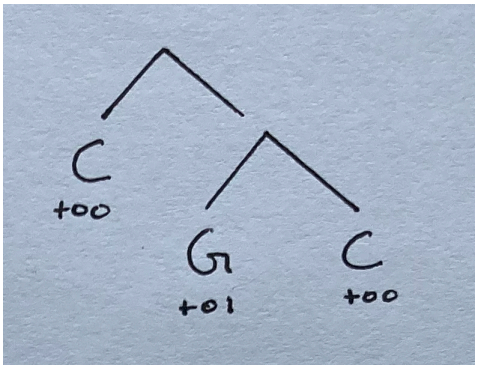

Figure 3. Model A Task 2 (Open-Composition) derivation, first example. Letters indicate *Stufen* root names, numbers indicate *c5* value. Model A demonstrates *Ursatz* form.

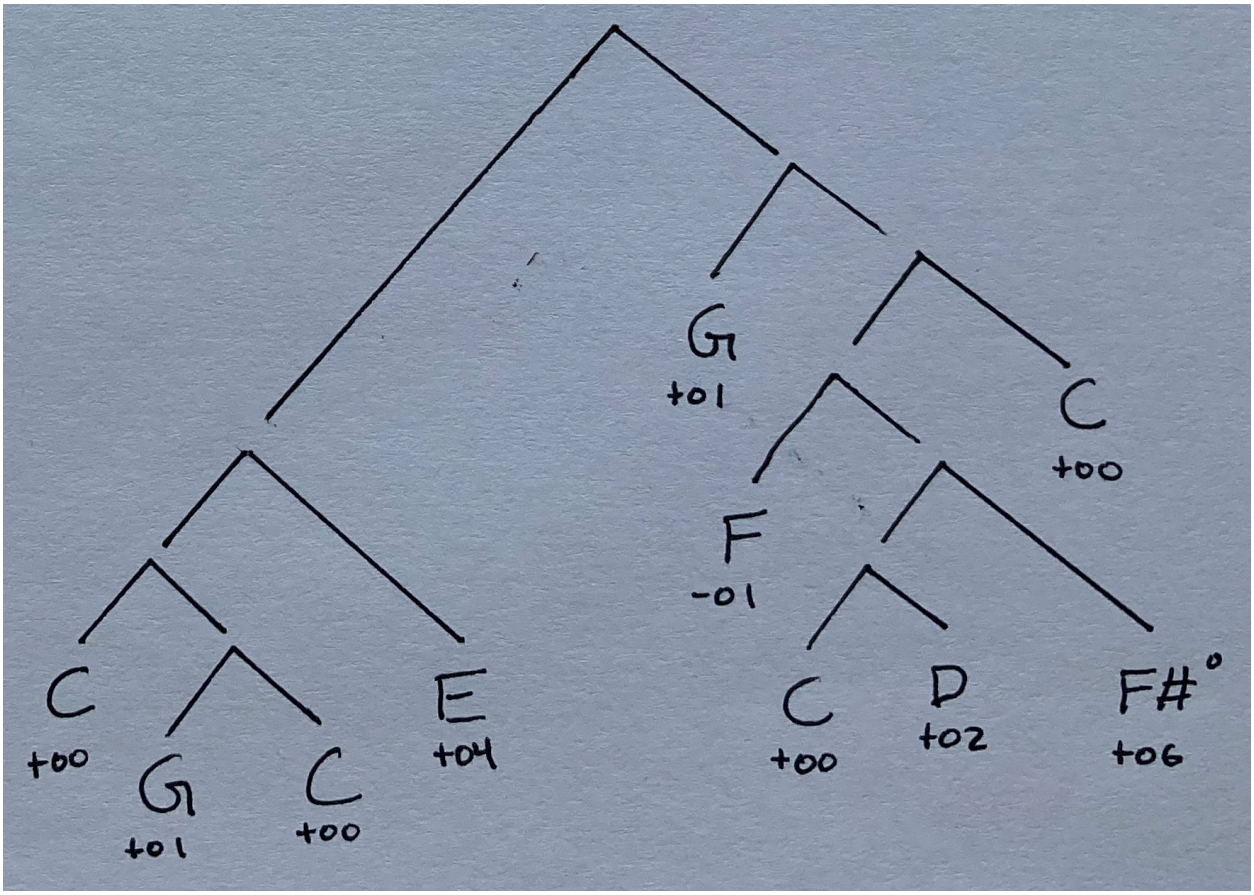

Figure 4. Model A Task 2 (Open-Composition) derivation, second example. Letters indicate *Stufen* root names; lowercase indicates minor, º indicates diminished, and capital indicates major types. Numbers indicate *c5* value. Model A overgenerates musical pieces; this example has both a surface ordering and a deep harmonic structure that disagrees with expectations from music theory.

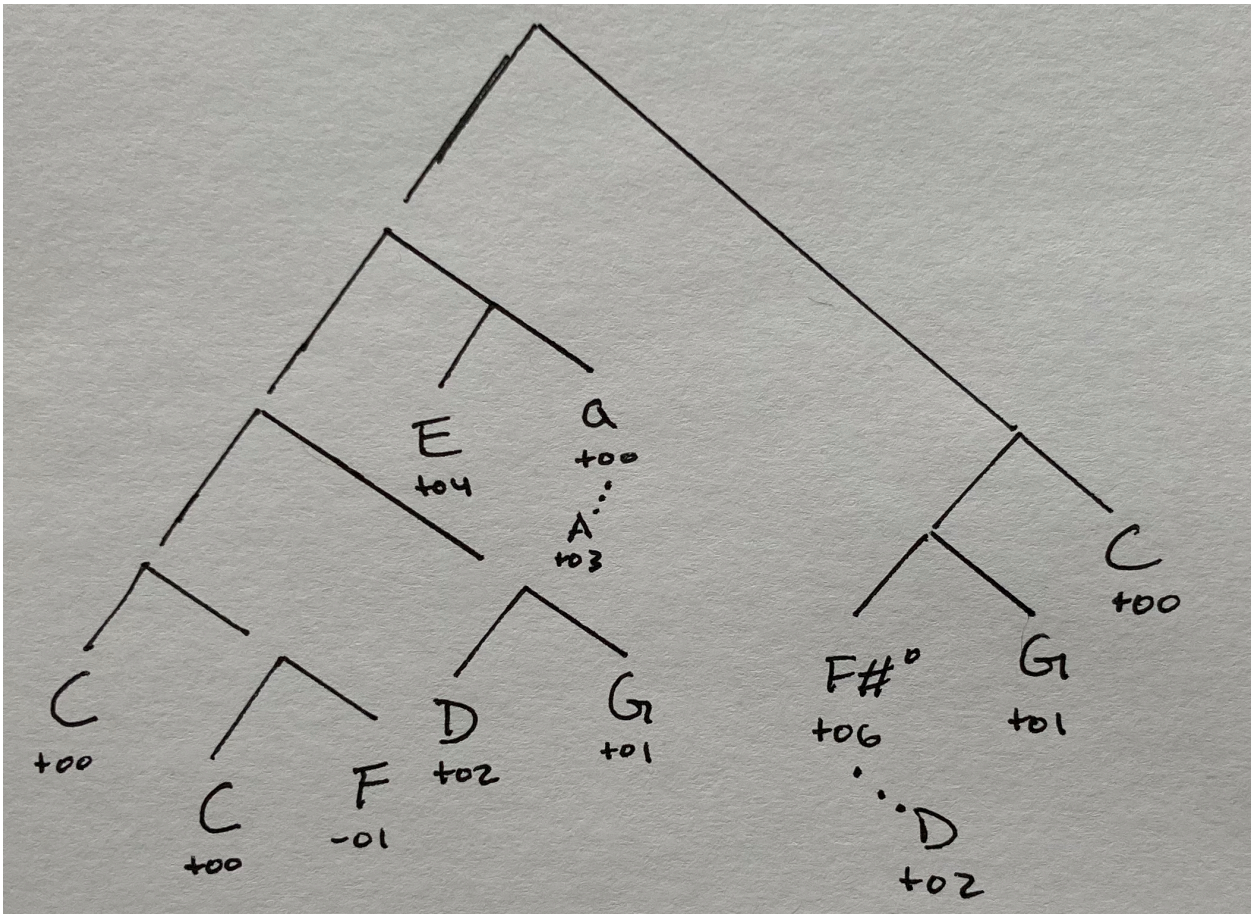

Figure 5. Model B, Task 1 (Tebe-Composition) derivation. Letters indicate *Stufen* root names; lowercase indicates minor, º indicates diminished, and capital indicates major types. Numbers indicate *c5* value. Model B is able to derive both the surface chord ordering of *Tebe Poem* and a sensible deep harmonic structure.

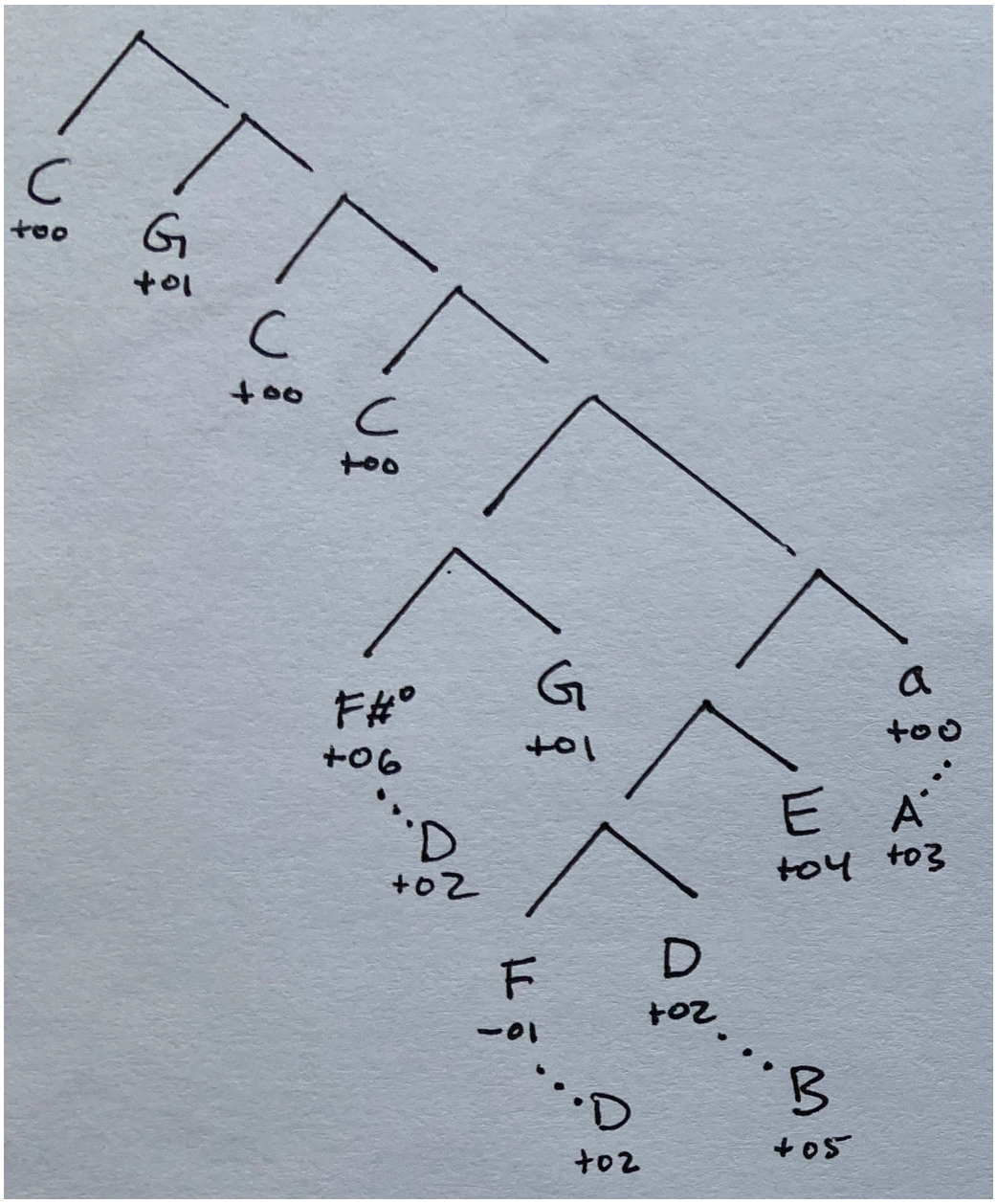

Figure 6. Model B, Task 2 (Open-Composition) derivation, first example. Letters indicate *Stufen* root names; lowercase indicates minor, º indicates diminished, and capital indicates major types. Numbers indicate *c5* value under nodes with solid lines. Nodes with dotted lines indicate use of *c3* value by parent *Stufe* (e.g. F Major used *c3* = +02 to Merge with D).

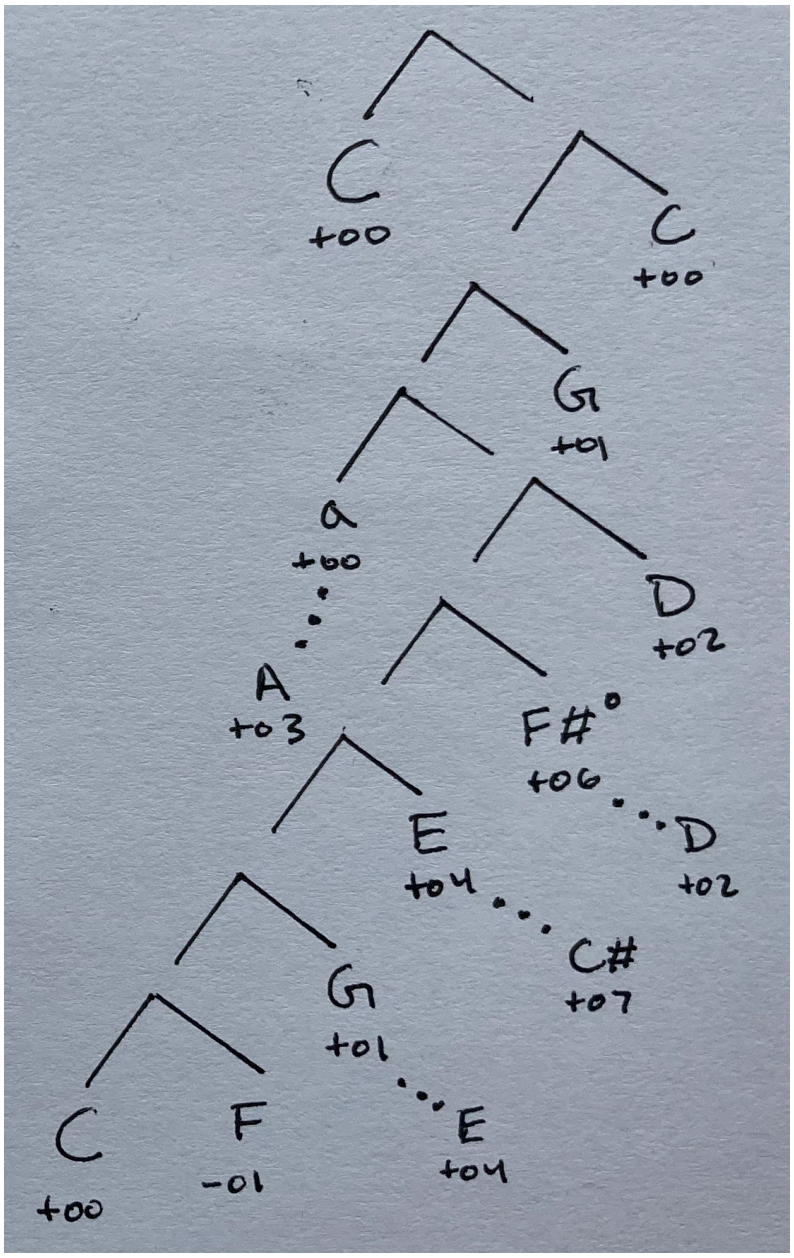

Figure 7. Model B, Task 2 (Open-Composition) derivation, second example. Letters indicate *Stufen* root names; lowercase indicates minor, º indicates diminished, and capital indicates major types. Numbers indicate *c5* value under nodes with solid lines. Nodes with dotted lines indicate use of *c3* value by parent *Stufe*.

Appendix: Model Code

Model A

```
import random


class Stufe:
    """Musical equivalent of "Lexical Item"
    Chromatic edition"""
    # Circle of Fifths in sharps, octave equivalence
    FIFTHS_NAMES = ['C', 'G', 'D', 'A', 'E', 'B',
                    'F#', 'C#', 'G#', 'D#', 'A#', 'F']
    FIFTHS_NAMES_MINOR = ['a', 'e', 'b', 'f#', 'c#', 'g#',
                          'd#', 'a#', 'f', 'c', 'g', 'd']

    def __init__(self, c5=0, major=True, dim=False):
        # default ctor
        self.is_major = major
        # setting to Major overrides diminished
        self.is_dim = dim if not major else False
        # circle of fifths value
        self.c5 = c5
        # circle of thirds value represents the c5 value
        # that this chord would have after a "covert progression"
        # (Mukherji, 2014: 358) down a minor third.
        if self.is_dim:
            self.c3 = (c5 + 8) % 12
        else:
            # minor and major have same c3
            self.c3 = (c5 + 3) % 12

        self.name = self.get_name()

    def get_name(self):
        # should work for negative cf too
        if self.is_major:
            return self.FIFTHS_NAMES[self.c5 % 12]
        elif self.is_dim:
            return self.FIFTHS_NAMES[self.c5 % 12] + "-dim"
        else:
            return self.FIFTHS_NAMES_MINOR[self.c5 % 12]

    def __str__(self):
        return f"{self.name}: c5 = {self.c5}"


class SyntacticObject:
    def __init__(self, m1, m2):
        """
        Represents the output of Merge.

        :param m1: a Stufe or SyntacticObject
        :param m2: a Stufe or SyntacticObject
        """

        self.items = (m1, m2)
```

```
        # Latter object projects syntactic features
        self.c5 = m2.c5
        self.c3 = m2.c3

    def __str__(self):
        # recursively access contained stufen
        return f"[{str(self.items[0])}, {str(self.items[1])}]"

    def spell_out(self):
        """
        Returns a string of the surface chords in this SO.
        String does not represent tree structure.

        :return: str
        """
        return self._spell_out_helper(self)[:-1]

    def _spell_out_helper(self, so):
        """
        Recursively access string of surface chord
        :param so: SyntacticObject
        :return: str
        """
        if isinstance(so, Stufe):
            return so.name + ' '
        else:
            return self._spell_out_helper(so.items[0]) \
                   + self._spell_out_helper(so.items[1])

"""
class LexicalArray:
    # make Lexical Array explicit
    def __init__(self, items):

        default ctor
        :param items: collection of Stufe

        self.items = set(items)


class Workspace:
    # make Workspace explicit
    def __init__(self, items):

        default ctor
        :param items: collection of syntactic objects

        self.items = set(items)
"""


class Composer:
    """
    Model A:
    Stochastic Free-Merge composer dispensing with Agree, indexing,
    and internal Merge (transformations). SpellOuts SyntacticObjects
    that pass Ursatz Filter. For simplicity, derivations are completed
```

```
    in C Major/A minor (WLOG to other keys).
    """
    class Stage:
        # For Select to operate on, to be consistent with C&S 2011
        def __init__(self, la=set(), workspace=set()):
            """
            default ctor
            :param la: set of Stufe objs
            :param w: set of SyntacticObject and Stufe objs
            """
            # Lexical Array
            self.la = la
            # Workspace
            self.workspace = workspace

        def __str__(self):
            return f"<{str(self.la)}, {str(self.workspace)}>"

        def print(self):
            # print stage contents nicely
            print("<{", end='')
            # lexical array
            print(*self.la, sep=', ', end='}, {')
            # workspace
            print(*self.workspace, sep=', ', end='}>\n')

    def __init__(self):
        self.stage_i = 0

    def filter(self, so) -> bool:
        """
        Returns true if so is "fully interpretable." In this case,
        whether it exhibits the Ursatz at the root.

        :param so: SyntacticObject
        :return: bool
        """

        is_tonic = (so.c5 == 0)
        has_dominant = False
        starts_with_tonic = False

        if isinstance(so.items[1], SyntacticObject):
            has_dominant = (so.items[1].items[0].c5 == 1)

        if is_tonic and has_dominant:
            starts_with_tonic = self._filter_helper(so.items[0])

        return is_tonic and has_dominant and starts_with_tonic

    def _filter_helper(self, so) -> bool:
        """
        recursively check for tonic Stufe in left branches

        :param so: SyntacticObject or Stufe
        :return: bool
        """
        if isinstance(so, Stufe):
```

```
            return so.c5 == 0
        else:
            return self._filter_helper(so.items[0])

    def select(self, item: Stufe, stage: Stage) -> Stage:
        """
        Select as defined in C&S. Moves an item from LA into
        Workspace.
        R: item in Stage.la

        :param item: Stufe
        :param stage: Stage
        :return: Stage
        """
        stage.la = stage.la - {item}
        # Stufe doesn't have == overloaded, so workspace
        # will store distinct copies of otherwise equivalent stufen
        stage.workspace.add(item)
        return stage

    def select_random(self, stage: Stage) -> Stage:
        """
        Moves a random Stufe from the LexicalArray to
        the Workspace.
        R: not Stage.la.empty()

        :param stage: Composer.Stage
        :return: Composer.Stage
        """
        # oof TODO: consider not using hash tables
        item = random.choice(tuple(stage.la))
        return self.select(item, stage)

    def merge(self, so1, so2, stage: Stage) -> SyntacticObject:
        """
        Performs external Merge on two SO's in stage.workspace
        as defined in C&S.
        REQUIRES: s01, s02 in stage.workspace
        MODIFIES: stage

        :param s01: Stufe or SyntacticObject
        :param s02: Stufe or SyntacticObject
        :param stage: Composer.Stage
        :return: SyntacticObject
        """
        stage.workspace = ((stage.workspace - {so1}) - {so2})
        new_so = SyntacticObject(so1, so2)
        stage.workspace.add(new_so)
        return new_so

    def merge_random(self, stage: Stage) -> SyntacticObject:
        """
        Performs Merge on two random SO's in stage.workspace.

        :param stage: Stage
        :return: stage
        """
        # oof TODO: consider not using hash tables
```

```
        so1, so2 = random.sample(tuple(stage.workspace), 2)
        return self.merge(so1, so2, stage)

    def derive(self, la, verbose=True):
        """
        Executes a derivation starting with LexicalArray la.
        Every SO generated that passes Filter will be spelled out.

        :param la: collection of Stufe objs
        :return: collection of derivations, bool
        """

        if len(la) < 2:
            print("Error: You need more than 2 Stufen to compose")
            return list()

        # set up (select 2)
        derivations = list()
        current = Composer.Stage(la=set(la), workspace=set())
        current = self.select_random(current)
        current = self.select_random(current)
        self.stage_i = 2

        # derivation
        while len(current.la) > 0 or len(current.workspace) != 1:
            flip = random.choice([0,1])
            if flip and len(current.la) > 0:
                # Select
                current = self.select_random(current)
            elif not flip and len(current.workspace) != 1:
                # Merge
                new_so = self.merge_random(current)
                # Filter and spell out
                if self.filter(new_so):
                    # found a valid derivation!
                    derivations.append(new_so)

            self.stage_i += 1
            if verbose:
                print(f"Stage #{self.stage_i}:")
                print(current)
                print()

        # end of derivation
        if self.filter(list(current.workspace)[0]):  # awk
            print("Derivation finished")
            return derivations, True
        else:
            # derivation crashed
            print("Derivation crashed")
            return derivations, False


def tebe_search(model: Composer) -> (int, int, list):
    """
    Continuously generates surfaces until Tebe poem is found.
    :param model: Composer
    :return: SyntacticObject
```

```
    """
    # all stufen hypothesized to be in Bortniansky's Tebe Poem
    lexicon = [(0, True), (0, True), (-1, True),
               (2, True), (1, True), (4, True), (0, False),
               (6, True), (1, True), (0, True)]
    TEBE = "C C F D G E a F# G C"
    lexical_array = list()

    for c5, is_major in lexicon:
        lexical_array.append(Stufe(c5=c5, major=is_major))

    #all_derivations = list()
    spelled = list()
    count = 0

    # check for tebe, derive again if necessary
    while TEBE not in spelled:
        count += 1
        new, success = model.derive(lexical_array, verbose=False)
        spelled = [ d.spell_out() for d in new ]
        #all_derivations.extend(new)

    return spelled, count, new


def main():
    # tebe testing
    # all stufen hypothesized to be in Bortniansky's Tebe Poem
    lexicon = [(0, True), (0, True), (-1, True),
               (2, True), (1, True), (4, True), (0, False),
               (6, True), (1, True), (0, True)]
    lexical_array = list()

    for c5, is_major in lexicon:
        lexical_array.append(Stufe(c5=c5, major=is_major))

    model = Composer()
    derivations, success = model.derive(lexical_array)

    print("All Derivations\n===============")
    print(derivations)
    if len(derivations) > 0:
        print(derivations[-1].spell_out())

    print("\nSearch for tebe\n===============")
    model = Composer()
    completed, count, so_list = tebe_search(model)
    print("Search for tebe finished")
    print(f"Found surface: {completed}\nafter {count} attempts\n")
    print([ str(so) for so in so_list ])

    return 0


if __name__ == "__main__":
    main()
```

Model B

```
import random
import itertools

from model import Composer, Stufe, SyntacticObject


class ComposerB(Composer):
    # Mukherji (2014) model
    def agree(self, so1, so2) -> bool:
        """
        Stufen-based Agree. Returns whether so1 and
        so2 are Merge-able in the order specified.

        :param so1: Stufe or SyntacticObject
        :param so2: Stufe or SyntacticObject
        :return: bool
        """

        # Western Tonal music parameterization
        do_agree = (0 <= so1.c5 - so2.c5 <= 1) or (0 <= so1.c3 - so2.c5 <= 1)
        # relative major clause for minor Stufen
        if not do_agree and isinstance(so2, Stufe):
            do_agree = not so2.is_major and not so2.is_dim \
                       and (0 <= so1.c5 - so2.c3 <= 1)
        return do_agree

    def get_mergables(self, stage: Composer.Stage) -> (bool, list):
        """
        Checks if a merge is possible with items in stage.workspace.
        Returns false if no possible merges exist. Returns a list of
        SO pairs if true.

        :param stage: Composer.Stage
        :return: bool, list of tuples
        """

        success = False
        merges_possible = list()
        # order matters
        for so1, so2 in itertools.permutations(stage.workspace, r=2):
            if self.agree(so1, so2):
                # TODO: stop at first pair found?
                success = True
                merges_possible.append( (so1, so2) )

        return success, merges_possible

    def derive(self, la, verbose=True):
        """
        Executes a derivation starting with Lexical Array la.
        Flips a coin to decide whether to Select or to Merge.
        If Merging, merges agreeing SO's if possible, otherwise
        makes no operation. Every SO generated that passes Filter
        will be spelled out.

        :param la: collection of Stufe objs
```

```
        :param verbose: bool
        :return: collection of derivations, bool
        """

        if len(la) < 2:
            print("Error: You need more than 2 Stufen to compose")
            return list()

        # set up (select 2)
        derivations = list()
        current = Composer.Stage(la=set(la), workspace=set())
        current = self.select_random(current)
        current = self.select_random(current)
        self.stage_i = 2

        # derivation
        while len(current.la) > 0 or len(current.workspace) != 1:
            flip = random.choice([0,1])
            if flip and len(current.la) > 0:
                # Select
                current = self.select_random(current)
            elif not flip and len(current.workspace) != 1:
                # Merge
                merge_possible, mergeables = self.get_mergables(current)
                if merge_possible:
                    so1, so2 = random.choice(mergeables)
                    new_so = self.merge(so1, so2, current)
                    # Filter and spell out
                    if self.filter(new_so):
                        # found a valid derivation!
                        derivations.append(new_so)
                else:
                    # crash clause
                    if len(current.la) == 0:
                        break

            self.stage_i += 1
            if verbose:
                print(f"Stage #{self.stage_i}:")
                current.print()
                print()

        # end of derivation
        if len(current.workspace) > 1 or not self.filter(list(current.workspace)[0]):
# awk
            # derivation crashed
            print("Derivation crashed")
            return derivations, False
        else:
            print("Derivation finished")
            return derivations, True


def tebe_search(model: ComposerB) -> (int, int, list):
    """
    Continuously generates surfaces until Tebe poem is found.
    :param model: Composer
    :return: SyntacticObject
```

```
    """
    # all stufen hypothesized to be in Bortniansky's Tebe Poem
    lexicon = [(0, True, False), (0, True, False), (-1, True, False),
               (2, True, False), (1, True, False), (4, True, False), (0, False,
False),
               (6, False, True), (1, True, False), (0, True, False)]
    TEBE = "C C F D G E a F#-dim G C"
    lexical_array = list()

    for c5, is_major, is_dim in lexicon:
        lexical_array.append(Stufe(c5=c5, major=is_major, dim=is_dim))

    #all_derivations = list()
    spelled = list()
    count = 0

    # check for tebe, derive again if necessary
    while TEBE not in spelled:
        count += 1
        new, success = model.derive(lexical_array, verbose=False)
        spelled = [ d.spell_out() for d in new ]
        #all_derivations.extend(new)

    return spelled, count, new


def main():
    # tebe testing
    # all stufen hypothesized to be in Bortniansky's Tebe Poem
    lexicon = [(0, True, False), (0, True, False), (-1, True, False),
               (2, True, False), (1, True, False), (4, True, False), (0, False,
False),
               (6, False, True), (1, True, False), (0, True, False)]
    lexical_array = list()

    for c5, is_major, is_dim in lexicon:
        lexical_array.append(Stufe(c5=c5, major=is_major, dim=is_dim))

    model = ComposerB()
    derivations, success = model.derive(lexical_array)

    print("All Derivations\n===============")
    print(derivations)
    if len(derivations) > 0:
        print(derivations[-1].spell_out())


    print("\nSearch for tebe\n===============")
    model = ComposerB()
    completed, count, so_list = tebe_search(model)
    print("Search for tebe finished")
    print(f"Found surface: {completed}\nafter {count} attempts\n")
    print([str(so) for so in so_list])

    return 0


if __name__ == "__main__":
```

```
main()
```