A proposal for automatic harmony analysis with Minimalist syntax

Sean P. Anderson
Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, Michigan, USA
seanpaul@umich.edu

Abstract

Recent work on the Identity Thesis for Music and Language (Katz and Pesetsky 2011; Mukherji 2014) provides fertile ground for formalizing and simulating computational, process-level models of musical composition. Within the Minimalist Program (Chomsky 2014), Anderson (2020) simulated a generative model of tonal harmony and structure. However in open-composition tasks the model fails to accurately capture Western tonal harmony. This is likely because of the model's reliance on hand-designed rules. We propose to learn the core Agree function, which determines when Merge applies, as a symbolic program by optimizing over the J.S. Bach Chorales dataset. A more accurate Minimalist harmony model would be valuable to music-language identity research. Additionally, a successful model could analyze harmonic structure and assist artificial composition systems via dataset augmentation.

Motivation

The question of music-language identity has been debated for centuries (Mukherji 2014). Despite noting clear relationships between music and language, Lerdahl & Jackendoff (1983) notably claimed that music does not have a syntax. However, the Minimalist program in generative linguistics (Chomsky 2014) provides a new understanding of syntax as a derivational instead of a representational system. This new view of language encouraged further interest in music-language identity and the formulation of the Identity Thesis, stipulating on both the equivalence of music and linguistic theory and a shared computational system for music and language in the mind (Katz and Pesetsky 2011; Mukherji 2014; Anderson 2020). These new perspectives on the presence of syntax in music rely on the Minimalist idea of repeated application of an operation Merge, which recursively builds syntax trees out of lexical items (Katz and Pesetsky 2011; Collins and Stabler 2016). Importantly, these lexical items are not the same between music and language, and therefore are not the focus of the above works.

Within this paradigm, (Katz and Pesetsky 2011; Rohrmeier 2007; Mukherji 2014) have each analyzed a portion of *Tebe Poem*, an 18th century choral piece written by Dimitri Bortniansky. These works asserted chords as lexical items upon which syntax operates. Anderson (2020) demonstrated that a simplified formalization of Minimalist syntax (Collins and Stabler 2016) coupled with syntactic features based on (Mukherji 2014) makes similar predictions on the hierarchical chord structure of *Tebe Poem* while improving in some aspects. Furthermore, Anderson (2020) formulate a distribution over harmonic orderings with hierarchies given an initial lexicon of chords. They sampled from this distribution in an Open-composition task, demonstrating the model's ability to generate both sensible chord prolongations and insensible ones.

The goal of such a model is to formulate a "type 2: Common properties of pieces within an idiom" (Katz and Pesetsky 2011) grammar. Any discrepancy between the grammar's assignments and the existence of human-composed pieces (within the idiom addressed, e.g. Western tonal music) would motivate improvements. The model in Anderson (2020) demonstrated clear discrepancies with Western Tonal harmony as a result of its reliance on hand-designed features.

The proposed project aims to improve on the tonal hierarchy model of (Anderson 2020) as a "type 2" grammar (Katz and Pesetsky 2011). The objective is to fit an Agree function to a corpus of human-composed chord progressions. Since the Agree function is in the form of a symbolic procedure, search for the best-fitting Agree function could be framed as a program synthesis problem (Gulwani et al. 2017).

A successful project would make the following contributions:

1. improve upon syntactic models of tonal harmony.
2. work towards better testing grounds for music-language identity theses.
3. provide richer data augmentation for the benefit of statistical/neural models of tonal composition.

The recent years have seen a plethora of increasingly successful music composition systems. For instance, the use of transformers has increased neural approaches' abilities to capture long-term structure (Huang et al. 2018). One of the best performing neural models in the J.S. Bach Chorales dataset is TonicNet (Peracha 2020). Peracha (2021) demonstrate that neural models can be improved by training on augmented datasets. A successful model with our approach could provide hiearchical descriptions of harmonic activity of chorales in the training set. It is possible that neural models of tonal composition, trained with this richer set of infor-

mation, could more effectively capture harmonic relationships in tonal music.

Methodology

We plan to take an existing Minimalist model of tonal harmony (Anderson 2020) and replace the Agree function with a program sketch. For the reader's convenience, core portions of the model are described below.

Dataset

We plan to use the J.S. Bach Chorales dataset, available at (https://github.com/czhuang/JSB-Chorales-dataset) (Huang et al. 2019). This dataset includes a piano-roll representation of 300+ chorales, but does not include chord annotations. Following (Peracha 2020), we can generate chord annotations using the music21 toolkit (Cuthbert and Ariza 2010). Since our model works with harmony only, the dataset for our model is just the resulting chord sequences.

Lexicon

Definition (lexicon)**.** A lexicon is a finite set of *Stufen*.

The Anderson (2020) model assumes that tonal music makes use of a lexicon containing Schenker's *Stufen* (pl.) A *Stufe* is an abstract entity behind the scale and chords of a particular key (Schenker 1973, 133-153). For our purposes it is simply a numerical object:

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, 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"}. These are discussed in more detail below.

c3 and c5 serve as syntactic features of a *Stufe*, and are uniquely defined by the root and type of the object. We use brackets to denote "accessing" a feature value from a *Stufe* object. For example, if C = <+00, +03, "C", "Major">, C[c5] = +00. Following (Mukherji 2014), each *Stufe* acquires its c5 features by its location in the Circle of Fifths (Mukherji 2014) (Figure 1). The c5 features enable the common harmonic progression V-I observed in tonal music. For instance in Western tonal music, adjacent *Stufen* could be ordered in the negative direction (e.g. G = +01 to C = +00) based on parameterization of Agree (defined below). 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) are instantiations of the same *Stufe*, one of type "Major" (Mukherji 2014).

Definition (c5)**.** c5, called a circle of fifths feature, is an integer in the interval [-12, +12], which corresponds to a location on the Circle of Fifths. We use "+00" to refer to the tonic.

Since many chord progressions are not simple traversals of the Circle of Fifths, we include a c3 feature. For instance, in the C Major scale, the F Major triad often functions as a IV chord leading to a V chord. This progression could be made legal by a covert, or invisible, progression from F *Stufe* to a d *Stufe* via a thirds-based relationship (Mukherji 2014). Looking at the Circle of Fifths, d has a c5 value of +02, which could lead to a G *Stufe*. Notably F-d-G (functionally: IV-ii-V) does occur in Western Tonal music.

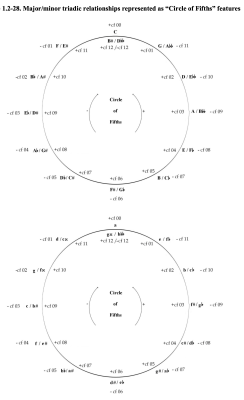

Figure 1: Circle of Fifths with c5 features, reprinted with permission from Mukherji (2014, 338). Each *Stufe* is related to others by intervals of a perfect fifth. Top: Major *Stufen*, Bottom: minor *Stufen*. Diminished *Stufen* and c3 values not pictured.

Definition (c3)**.** c3, or circle of thirds feature, is an integer value in the interval [-12, +12]. If a *Stufe* S is of type "Major," S[c3] = $(c5 + 3) \mod 12$, or the c5 value the *Stufe* would have if its root was a minor third below its own. If S is of type "minor," S[c3] = $(c5 + 3) \mod 12$ also, which is the c5 value of its parallel major *Stufe*. If S is of type "diminished," S[c3] = $(c5 + 8) \mod 12$, or the c5 value *Stufe* would have if its root was a major third below its own.

Typically, diminished chords including $F\#^o$ 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, 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 c3 definition for diminished *Stufen* is a significant simplification of the harmonic nature of these chords; a more accurate feature space would probably be required for success in this problem.

Generating harmonic analyses

We refer the reader to (Anderson 2020) for a full description of the model. Below we define the core operations Select, Merge, Filter, and our program sketch for Agree (Collins and Stabler 2016).

Definition (Select)**.** Let S be a stage in a derivation S = <LA, W>. If lexical token A $\in$ LA, then Select(A, S) = <LA - {A}, W $\cup$ {A}>.

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

Importantly, the result of Merge is itself a Syntactic Object (SO). For this project Merge can operate on any two distinct Syntactic Objects, which includes items "contained" (in Collins and Stabler (2016, 46)) within other items to be merged together, resulting in "Internal Merge."

Definition (Agree*)**.** Operation Agree(SO1, SO2) operates on Syntactic Objects SO1 and SO2. Agree is a program sketch, or a program with holes that need to be filled by a search algorithm. Agree(SO1, SO2) takes the following form:

$$\phi_1 \vee \phi_2 \vee ... \vee \phi_n$$

where ϕ_i is a boolean formula of the form:

Labels(SO1)[c#] - Labels(SO2)[c#] == k

where $\# \in \{3, 5\}$ and $k \in \{-1, 0, 1\}$. n is a hyperparameter that will be additionally investigated.

Labels returns the SO of the right-most leaf of the input tree (Anderson 2020). Agree determines when Merge can apply. $\#$ and k denote holes that will be paramerized during program synthesis.

To determine if a derivation is a viable harmonic prolongation, we use the Filter operation:

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] = LeftLeaf(SO)[c5],

(ii) Label(J)[c5] = Label(SO)[c5] + 1

Filter(SO) = False otherwise.

LeftLeaf returns the label of the left-most leaf in the SO tree (Anderson 2020). Filter simply 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 and Jackendoff 1983; Katz and Pesetsky 2011).

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

1. Instantiate a derivation by loading all chord objects under consideration (the lexicon)
2. Select *Stufen* randomly into the workspace (i.e. asserting they will appear in resulting progression).
3. Over all syntactic object pairs that satisfy Agree, pick ones at random and apply Merge.
4. When there is one SO remaining, apply Filter. If Filter returns true, then the derivation succeeds, and the chord ordering is the SO's leaves read from left to right. If not, the derivation crashes and we start over.

Searching for an Agree function

Searching for an Agree function (given n) would amount to searching over parameterizations of the Agree sketch. To surmount the computational costs of enumerative search, an appropriate algorithm needs to be selected, such as those used in program synthesis (Gulwani et al. 2017). The specification for a synthesis iteration could come in the form of input-output examples, where the input is a set of chords in a phrase from one of J.S. Bach's chorales, and the output is the correct ordering of those chords.

However, synthesis may be difficult since 1) the dataset includes only positive examples, and 2) checking whether a chord ordering would be valid according to a completed Agree function is not straightforward. One strategy is to execute Derive(LA) on the set of *Stufen* present in the ordering for many iterations until the correct ordering appears (demonstrating its validity) or a cutoff is reached (suggesting the ordering is invalid and the Agree function fails). This could be inefficient. Another is to parse the ordering directly using the Agree function, which requires inverting the model. Additionally, c3 and c5 features will likely not be enough to accurately depict harmonic prolongations present in the J.S. Bach Chorales. Further exploration of other features, for instance crafting Circle of 2nds, 4ths, 6ths and 7ths features, may be necessary.

Evaluation A resulting parameterization of Agree will determine a set of valid chord progressions. Evaluation of a resulting algorithm would include the proportion of chord orderings in J.S. Bach Chorales that are contained in the resulting set. Additionally, examination of the chord orderings deemed valid and outside of the Chorales would be qualitatively useful.

References

Anderson, S. 2020. A linguistic model of minimalist syntax composes tebe poem.

Chomsky, N. 2014. *The minimalist program*. MIT press.

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

Cuthbert, M. S., and Ariza, C. 2010. music21: A toolkit for computer-aided musicology and symbolic music data.

Gulwani, S.; Polozov, O.; Singh, R.; et al. 2017. Program synthesis. *Foundations and Trends® in Programming Languages* 4(1-2):1–119.

Huang, C.-Z. A.; Vaswani, A.; Uszkoreit, J.; Shazeer, N.; Simon, I.; Hawthorne, C.; Dai, A. M.; Hoffman, M. D.; Dinculescu, M.; and Eck, D. 2018. Music transformer. *arXiv preprint arXiv:1809.04281*.

Huang, C.-Z. A.; Cooijmans, T.; Roberts, A.; Courville, A.; and Eck, D. 2019. Counterpoint by convolution. *arXiv preprint arXiv:1903.07227*.

Katz, J., and Pesetsky, D. 2011. The identity thesis for language and music. *URL http://ling. auf. net/lingBuzz/000959*.

Lerdahl, F., and Jackendoff, R. 1983. *A generative theory of tonal music*. MIT Press.

Mukherji, S. 2014. Generative musical grammar-a minimalist approach. *PhD Diss. Princeton University*.

Peracha, O. 2020. Improving polyphonic music models with feature-rich encoding. In *Proceedings of the 21st Int. Society for Music Information Retrieval Conference*.

Peracha, O. 2021. Js fake chorales: a synthetic dataset of polyphonic music with human annotation. *arXiv preprint arXiv:2107.10388*.

Rohrmeier, M. 2007. Modelling dynamics of key induction in harmony progressions. In *Proceedings of the 4th Sound and Music Computing Conference*, 82–89.

Schenker, H. 1973. *Harmony*. Cambridge: The MIT Press.