Composing with Kulitta

Donya Quick
Yale University Department of Computer Science
donya.quick@yale.edu

ABSTRACT

Kulitta is a Haskell-based framework for automated and algorithmic music composition. Kulitta can be used to generate anything from short phrases to complete pieces of music with little user input, or the system can be used as a toolbox for creating algorithmic compositions with far more human composer involvement and control. Kulitta's modularity facilitates the system's use in many different ways as part of a larger algorithmic composing workflow.

1. INTRODUCTION

This paper presents an overview of a new automated composition system, called Kulitta[1], and some of the ways in which it can be used to create new pieces of music. Kulitta can produce a wide variety of styles and is highly customizable.

Algorithms for music composition can be used for creating music in an automated manner or as tools that enable human composers to create rich and complex works. Algorithms for generating music can produce a wide range of results, ranging from highly mathematical and modern to very traditional. However, algorithms for emulating human decision making in music often face a harder task than those for simply creating interesting auditory experiences. This is because many constrained musical tasks are intractable if approached directly on a large scale—the solution spaces are gigantic even for a machine. For example, even the simple task of octave assignment for n pitch classes on an 88-key piano has an exponential growth pattern: there are at least n^7 solutions, since there are at least 7 octaves for each pitch class. This solution space will quickly grow out of control for anything but trivially small values of n. Kulitta avoids this problem in part by utilizing the principal of *musical abstraction*.

Music has many levels of abstraction. Concepts like Roman numerals and chord function are abstract because they can be interpreted in many different ways. Kulitta works with music at multiple levels of abstraction and composes by iteratively fleshing out an idea until the desired level of detail is reached.

As illustrated in Figure 1, Kulitta's components, or modules, fall into three main categories:

1. **Abstract/structural generation**: a collection of Schenkerian-inspired models and algorithms for iteratively generating harmonic structure or other abstract, hierarchical musical features. In general, output at this level is not yet music and requires additional detail before it can be performed/heard.
2. **Musical interpretation**: mathematical models and constraint satisfaction algorithms for turning abstract musical features into more detailed music. Much of the determination of musical style takes place at this level.
3. **Learning**: offline learning algorithms to derive production probabilities for musical grammars, which can then be used to generate abstract structure.

Kulitta is superficially similar to two other well-known systems for automated composition: David Cope's EMI [2] and Kemal Ebcioğlu's expert system for synthesizing chorales [3]. All have the ability to learn features from a corpus of music and to examine music from different perspectives. However, Kulitta differs from the other two systems both in the particular algorithms used and their separability. One of the goals when creating Kulitta was to obtain a highly modular system that could both be configured as a stand-alone entity for automated composition and taken apart to be used like building blocks in more diverse systems for algorithmic composing. The system's components can be rearranged and repurposed as the user sees fit. When used as a toolbox for a human composer, Kulitta can be used in combination with other libraries by coding in a text editor. In this way, particularly when used in combination with other libraries, Kulitta is somewhat similar to other text-based algorithmic composing environments, such as Slippery Chicken [4] and SuperCollider [5].

Kulitta is implemented in Haskell [6], a functional programming language that lends itself to concise yet versatile code. Kulitta also uses the Euterpea library [7], which contains various musical representations for Haskell as well as support for real time MIDI I/O and writing MIDI files. Euterpea's musical representations for Haskell are particularly useful for defining style-specific algorithms for creating musical foregrounds (surface-level rhythmic and melodic details over a more abstract harmonic backbone). Scores in this paper were created by visualizing Kulitta's output (MIDI files) with Musescore [8].

This paper provides a high-level overview of Kulitta's features and ways in which Kulitta can be used for algorithmic and automated composition. For more detail on all of

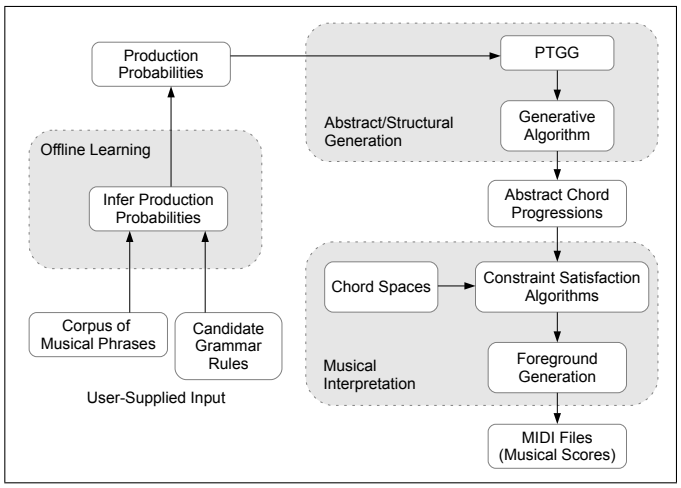

Figure 1. The main components of Kulitta arranged to automatically generate musical scores using a linear workflow with little user interaction during the generative process.

Kulitta's features and algorithms, and for more detailed examples of the generative process, readers are directed to the author's dissertation[1]. Kulitta's complete source code is also available online [1] .

2. MUSICAL GRAMMARS

The idea that music can be modeled like spoken language has become increasingly accepted in music theory through works such as Generative Theory of Tonal Music [9]. Grammars have been explored both generatively and analytically in music. Uses of musical grammars include algorithmic composition [10, 11] and modeling aspects of music such as harmony [12, 13] and jazz leads [14].

Kulitta features a category of grammars called Probabilistic Temporal Graph Grammars[15, 16], or PTGGs. PTGGs are similar to more typical context free grammars, or CFGs, but PTGGs allow the simultaneous generation of harmonic and metrical structure through the use of a parameterized alphabet. Table 1 shows an example of a PTGG that generates abstract harmonic and metrical structure in this way. Harmonic symbols in a PTGG are parameterized with duration, and rules are functions on that duration. This allows CFG-like rules to operate over an alphabet that is actually infinite. PTGGs also support repetition through variable instantiation (*let-in* statements), a prominent feature of music that is easily modeled using programming languages-like features but that is often impossible to model in more typical CFGs. Several musical PTGGs are built into Kulitta, such as those shown in Tables 1 and 2, but users can also specify their own.

Kulitta also has support for learning production probabilities for PTGGs and other musical CFGs given a corpus of analyzed data. Kulitta's learning module uses an extended, oracle-based version of the traditional inside-outside algorithm for learning CFG production probabilities. Figure 2 shows an example of a phrase produced using the grammar in Table 1, the production probabilities for which were learned from Bach chorales.

0.41	$T^t \rightarrow T^t$	0.64	$D^t \rightarrow D^t$
0.30	$T^t \rightarrow T^{t/2}T^{t/2}$	0.09	$D^t \rightarrow D^{t/2}D^{t/2}$
0.16	$T^t \rightarrow D^{t/2}T^{t/2}$	0.27	$D^t \rightarrow S^{t/2}D^{t/2}$
0.12	$T^t \rightarrow T^{t/2}D^{t/2}$	0.95	$S^t \rightarrow S^t$
		0.05	$S^t \rightarrow S^{t/2}S^{t/2}$

Table 1. An example of a PTGG over chord function symbols, tonic (T), dominant (D), and subdominant (S), parameterized by duration as a superscript. Production probabilities shown were derived from a corpus of major Bach chorales [1].

0.2	$U_x^t \rightarrow U_x^t$	0.2	$D_x^t \rightarrow D_x^t$
0.2	$U_x^t \rightarrow U_{\lvert x-1\rvert}^{t/2}\ U_{x+1}^{t/2}$	0.2	$D_x^t \rightarrow D_{\lvert x-1\rvert}^{t/2}\ D_{x+1}^{t/2}$
0.2	$U_x^t \rightarrow U_{\lvert x-1\rvert}^{t/2}\ D_{x+1}^{t/2}$	0.2	$D_x^t \rightarrow D_{\lvert x-1\rvert}^{t/2}\ U_{x+1}^{t/2}$
0.2	$U_x^t \rightarrow U_{x+1}^{t/2}\ U_{\lvert x-1\rvert}^{t/2}$	0.2	$D_x^t \rightarrow D_{x+1}^{t/2}\ D_{\lvert x-1\rvert}^{t/2}$
0.2	$U_x^t \rightarrow U_{x+1}^{t/2}\ D_{\lvert x-1\rvert}^{t/2}$	0.2	$D_x^t \rightarrow D_{x+1}^{t/2}\ U_{\lvert x-1\rvert}^{t/2}$

Table 2. A non-harmony-based PTGG for melodic motion, where U means "up" and D means "down." This PTGG uses uniform production probabilities and stores two pieces of information as parameters: the current duration as a superscript and the amount of movement in some tonal system as a subscript (for example, Figure 3 interprets the values as indices into a pentatonic scale). When interpreted as motion within a particular tonal system, this grammar produces interesting melodic contours.

3. CHORD SPACES

Chord spaces are mathematical constructs that group chords or other musical features in musically meaningful ways. To construct a chord space, a set of chords is partitioned using an *equivalence relation* [2] . A group of inter-related chords belong to the same *equivalence class*. Pitches are represented as numbers and chords are represented as vectors of pitches.

Several equivalence relations based on classical music theory are presented by Tymoczko [17] and Callender et al. [18]. These relations can be combined to model many different music theoretic concepts. For example, a relation called octave and permutation equivalence, or OP-equivalence, groups together chords that share the same sets of pitch classes. Under OP-equivalence, all C-major triads with a C, E, and G would belong to the same equivalence class. This chord space is particularly useful for tasks such as voice-leading assignment. Kulitta's chord space module provides implementations for a number of combinations of these equivalence relations in addition to another, new space called mode space [1].

Chord spaces allow one progression to be transformed into another through the use of path-finding algorithms and musical constraints [1, 19]. Reharmonization and free composition given an abstract harmonic "backbone" to follow are re-

[1] For Kulitta's complete source code and additional examples, please see the author's website: `www.donyaquick.com`

[2] An equivalence relation is a relation that is symmetric, transitive, and reflexive. An equivalence relation forms a partition over a set.

ducible to such a path finding problem. When using a chord space to perform either of these tasks, it is also possible to drastically alter the style of music by simply changing the chord space used or by performing subsequent reharmonization steps through additional chord spaces [1]. A series of Roman numerals can become anything from series of simple, traditional triads to a complex, modern-sounding progression with many voices.

4. FOREGROUND AND PERFORMANCE

Kulitta has built-in foreground algorithms for two different styles of music: chorales and jazz, as shown in Figures 2 and 4 respectively. Kulitta's jazz foreground algorithms can produce either a simple series of jazz chords with a bassline (two instruments or two hands on a piano) or a bossa nova with a melody, chords, and bassline. Perhaps of greater interest to composers is the fact that the user can, using Haskell and Euterpea, create his or her own foreground algorithm, interpreting abstract harmony as desired. Figure 5 was produced using a custom foreground algorithm for creating simple piano pieces for human performance.

Currently, when Kulitta's modules are used as-is with little or no customization, output is limited to that of a very plain musical score. Although the possibility exists for the user to create a new grammar or foreground algorithm that introduces more performance-oriented detail, Kulitta's built-in grammars and foreground algorithms only address pitch and duration for notes—additional features such as volume, tempo, and articulation are not included. However, as any musician will know, performance is an integral part of music, and it is important to consider that even a seemingly bland score from Kulitta can have many possible interpretations and realizations as sound.

5. USING KULITTA

Using either one of Kulitta's built-in foreground algorithms or a new, user-created one, Kulitta can be configured as a self-contained music generator that essentially allows the user to press a button and generate a new piece of music from a random number seed. This has primarily two uses: (1) generating new exercises as practice material for human musicians—for example, creating many similar but unique short phrases that exemplify a particular technique, and (2) testing musical hypotheses by examining many phrases or pieces generated from a particular model.

Many analytical models for music are never tested in a generative setting. Turning analytical models into generative ones is an easy way to test them for weaknesses, since it will clearly show what is and, often more importantly, is not captured by the model. This can help to confirm or reject existing music theoretic hypotheses and yield new directions for further research.

Although Kulitta can be autonomous, the framework can also be used as a set of tools for creating algorithmic compositions with more human involvement. When used this way, Kulitta becomes more like a composition partner, supplying ideas and material for the composer to further refine.

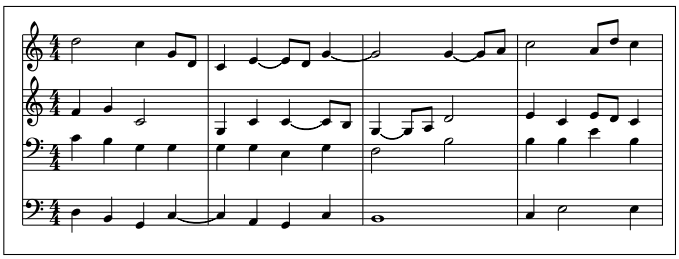

Figure 2. A chorale phrase generated by Kulitta using the PTGG shown in Table 1.

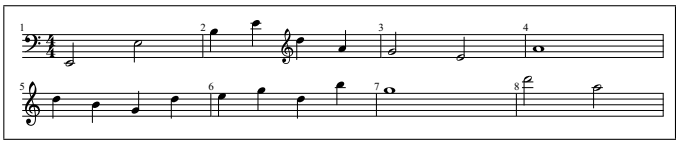

Figure 3. An 8-measure phrase generated using the PTGG from Table 2 and interpreted using a pentatonic scale.

Kulitta's support for PTGGs and chord spaces have great potential for user customization. Users can create new PTGGs and easily generate from them using Kulitta's generative algorithms for PTGGs. New chord spaces can also be defined to model other music theoretic concepts or they could be derived from a data set. Although Kulitta's built-in PTGGs and chord spaces are primarily harmony-centric, these features can be repurposed to model other elements of music. One can easily construct a PTGG over another alphabet, such as the up/down motion illustrated in Table 2, which was used to produce the single-line phrase in Figure 3. Similarly, chord spaces have the potential to model more than vertical harmony. The same approach can also be used to model horizontal features such as musical contours [20], which are also representable as vectors.

6. CONCLUSION

Kulitta is a powerful entity for automated music composition and tool for algorithmic composition. The system can be configured to generate music with little user input or it can be customized and extended as the user pleases. As shown in the examples here, there are many ways that Kulitta's modules can be used to create diverse and interesting music.

One of Kulitta's main limitations as an algorithmic composition tool is the requirement that users must be familiar with Haskell in order to adapt the system to their needs. Coding in Haskell is required to define new PTGGs, foreground algorithms, and so on. While these features can be created with relatively little code, Haskell can be somewhat daunting to novice programmers or those unfamiliar with functional programming languages. Intermediate levels of representation, implementations of Kulitta's modules in other programming languages, and other user interfaces to facilitate use by less experienced programmers are currently being explored.

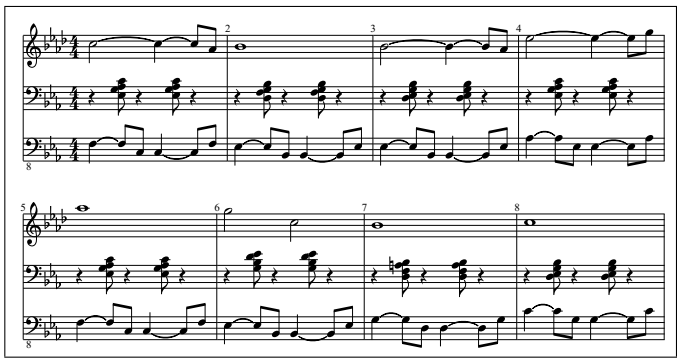

Figure 4. A bossa nova phrase generated by Kulitta.

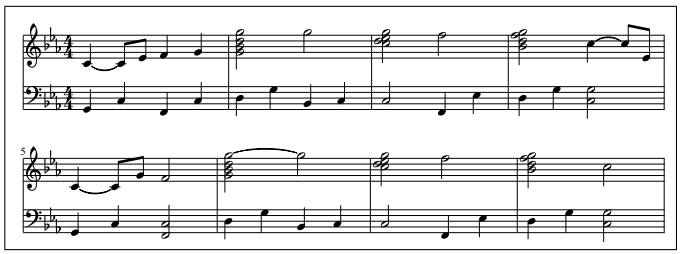

Figure 5. A phrase taken from a solo piano algorithmic composition created using Kulitta. The first and second half have the same abstract harmonic structure and differ only in surface-level detail.

Finally, the various compositional workflows discussed so far for using Kulitta are primarily linear. Multiple instances of the same module may be used in a particular setting (such as multiple chord spaces), but the output of one instance of a module will still generally feed into a different instance to address a new level of abstraction rather than loop back to address the same level of abstraction. The reason for this linearity is that Kulitta currently has no built-in support for back-tracking during generation. Error-detection and back-tracking support are areas of ongoing work in Kulitta's development.

Acknowledgements: this research was supported in part by NSF grants CCF-0811665 and SHF-1302327.

7. REFERENCES

[1] D. Quick, "Kulitta: a Framework for Automated Music Composition," Ph.D. dissertation, Yale University, 2014.

[2] D. Cope, "Computer Modeling of Musical Intelligence in EMI," *Computer Music Journal*, vol. 16, no. 2, pp. 69–83, 1992.

[3] K. Ebcioğlu, "An Expert System for Schenkerian Synthesis of Chorales in the Style of J.S. Bach," in *Proceedings of the International Computer Music Conference*, 1984, pp. 135–140.

[4] M. Edwards, "An Introduction to Slippery Chicken," in *Proceedings of the International Computer Music Conference*, 2012, pp. 349–356.

[5] J. McCartney, "Rethinking the Computer Music Language: SuperCollider," *Comput. Music J.*, vol. 26, no. 4, pp. 61–68, Dec. 2002.

[6] S. Peyton Jones, "The Haskell 98 Language and Libraries: the Revised Report," *Journal of Functional Programming*, vol. 13, no. 1, pp. 0–255, Jan 2003.

[7] P. Hudak *et al.* (2014) Euterpea. [Online]. Available: http://hackage.haskell.org/package/Euterpea

[8] T. Bonte, N. Froment, and W. Schweer. (2014) MuseScore. [Online]. Available: http://www.musescore.org/

[9] F. Lerdahl and R. S. Jackendoff, *A Generative Theory of Tonal Music*. The MIT Press, 1996.

[10] M. Gogins, "Score generation in Voice-Leading and Chord Spaces," in *Proceedings of the International Computer Music Conference*, 2006, pp. 593–600.

[11] P. Worth and S. Stepney, "Growing Music: Musical Interpretations of L-systems," *Applications on Evolutionary Computing*, pp. 535–540, 2005.

[12] M. Rohrmeier, "Towards a Generative Syntax of Tonal Harmony," *Journal of Mathematics and Music*, vol. 5, no. 1, pp. 35–53, 2011.

[13] T. Winograd, "Linguistics and the Computer Analysis of Tonal Harmony," *Journal of Music Theory*, vol. 12, no. 1, pp. 2–49, 1968.

[14] R. M. Keller and D. R. Morrison, "A Grammatical Approach to Automatic Improvisation," in *Sound and Music Computing Conference*, 2007, pp. 330–337.

[15] D. Quick and P. Hudak, "Grammar-Based Automated Music Composition in Haskell," in *Proceedings of the first ACM SIGPLAN workshop on Functional art, music, modeling, and design*, 2013, pp. 59–70.

[16] ——, "A Temporal Generative Graph Grammar for Harmonic and Metrical Structure," in *Proceedings of the International Computer Music Conference*, 2013.

[17] D. Tymoczko, "The Geometry of Musical Chords," *Science Magazine*, vol. 313, no. 5783, pp. 72–74, 2006.

[18] C. Callender, I. Quinn, and D. Tymoczko, "Generalized Voice-Leading Spaces," *Science Magazine*, vol. 320, no. 5874, pp. 346–348, 2008.

[19] D. Quick and P. Hudak, "Computing with Chord Spaces," in *Proceedings of the International Computer Music Conference*, 2012, pp. 433–440.

[20] R. D. Morris, *Composition With Pitch-Classes: A Theory of Compositional Design*. Yale University Press, 1987.