Demo: Counterpoint Analysis and Synthesis

John Leo
Halfaya Research
Bellevue, WA, USA
leo@halfaya.org

Abstract

We present a Haskell library to help analyze and synthesize musical counterpoint. The tool allows expression of generic constraints in a higher-level musical language which are translated to a lower level for use both to find rule violations in existing musical counterpoint and generate (using an SMT solver) new music satisfying the contraints. The tool is intended for use by musicians who need only have a basic knowledge of Haskell.

Keywords: Counterpoint, Haskell, SMT

1 Introduction

We demonstrate work in progress on a tool to assist in the analysis of synthesis of musical counterpoint. Since the mid-18th century, the composition of counterpoint has been guided by principles enunciated in Fux's *Gradus ad Parnassum* [Fux 1965], first published in 1725. Fux presents an increasingly sophisticated series of "species" (one note against one note, two notes against one note, etc.) along with rules governing intervals between notes and motion between intervals designed to ensure consonance and independence of voices. It is well documented [Mann 1987] that composers including Haydn, Mozart and Beethoven both studied and taught from this text, and its fundamentals continue to taught to music students today (for example [Aldwell and Cadwallader 2018; Kennan 1999]).

In previous work, Cong and Leo [Cong and Leo 2019] encode the rules of first-species (note against note) counterpoint as type constructors in the dependenty-typed language Agda. This enforces correct-by-construction counterpoint and allows use of the Agda typechecker to return errors with no additional effort required. On the down side, these errors can be difficult to interpret for those less familiar with type errors. Furthermore, encoding rules into constructors proved awkward for handling more complex species and more global constraints. One could separate the construction of the pure music from the constraints which can be added as a local or global predicate, but one may also wish to deliberately violate some of the rules of strict counterpoint, as composers often do in practice, and simply be informed of where the violations occur without being prohibited from incorporating them.

FARM '22,
.

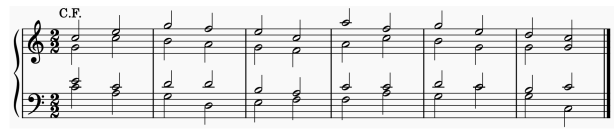

Figure 1. Beethoven Exercise 146

Another approach then is to write a special-purpose "type checker" which can be run on previously-created music and which can generate clear and precise error messages which are musically meaningful. The checker is ideally easily customizable in terms of what constraints one would like to impose. It turns out these constraints can be expressed in the quantifier-free logic of linear arithmetic and uninterpreted functions (QF-UFLIA [Barrett et al. 2010]), which allows one to not only analyze existing counterpoint, but also synthesize counterpoint satisfying the constraints using an SMT solver.

This tool, still a work in progress, has been implemented in Haskell and is available at [Leo 2022a]. Haskell was chosen as a high-quality and feature-rich SMT library called SBV [Erkök 2022] is available. It makes heavy use of functional programming, but dependent types are sometimes missed and the goal is to eventually port the work to the Agda Music Tools [Leo 2022b] library.

The following sections give an overview of current functionality. Future plans include integration with Liquid Haskell [Vazou et al. 2014] (for analysis) and Synquid [Polikarpova et al. 2016] (for synthesis). In addition to handling higher species we would like to also incorporate the rules and conventions of galant schemeta [Gjerdingen 2007], which informed much of the music composed in the era of Haydn and Mozart. Ideally one could use to tool to compose convincing music in the galant style.

Concurrently Cong [Cong 2022] is exploring another approach using a different representation of constraints within a type system.

2 An Example

We briefly describe one way in which this tool can be used for both analysis and synthesis. We first examine the example numbered 146 in the critical edition of Beethoven's studies with Haydn ([Ronge 2014]; see also [Nottebohm 1971], p. 31 and [Mann 1987], p. 115). It is shown in modern notation in Figure 1.

Figure 2. Three Notes Changed

Figure 3. Generated Counterpoint

Here Haydn has supplied the top voice, marked "C.F." for cantus firmus, and Beethoven's task was to supply the remaining four voices following first species counterpoint rules. He uses only tones from triads, all of them complete save the last and all in root position save the second triad in bar 4. Since the notes form triads they automatically satisfy consonance rules (only perfect intervals, thirds or sixths between any pair of notes vertically); note that a perfect fourth is prohibited in two part strict counterpoint but allowed with more voices. Perfect fourths can be found between the top two voices in the first, fourth and last bars.

There are boundary rules that the top and bottom voices must form octaves at the beginning and end, and motion rules which are designed to ensure independence of voices. In particular parallel and similar (both voices moving in the same direction) motion into fifths and octaves is prohibited. Here we can see Beethoven has in fact made two errors in the top two voices: similar motion into a fifth in bar 3 and then into an octave between bars 3 and 4. Haydn fixes both errors by changing the F in the the second voice to an A.

For simplicity we now focus only on the top two voices. Feeding Beethoven's notes into the tool as two part first species counterpoint, it easily finds and reports errors in the use of perfect fourths, missing octaves at the boundaries, and similar motion into fifths and octaves. We can disable the boundary rules as they are not relevant and relax the consance rule to allow perfect fourths, leaving only the motion errors. We can replace the note F that Haydn fixed with a hole, and run the tool in synthesis mode using the same rules. It makes the same fix Haydn did. Perhaps we gave it too much guidance, so instead we can three holes in a row near the error, which generates the solution shown in Figure 2. Note that this introduces a series of five parallel 6ths, and there is a "soft" rule that one should not have long sequences of 3rds and 6ths as that also limits independence of voices. However we can easily add a constraint limiting parallel intervals to at most two in a row; the solution returned changes the G at the start of bar four to an A, breaking the chain with a fifth, a small improvement but one might hope for more.

This can be done by adding a constraint to increase contrary motion (in which both voices move in opposite directions), and in fact we can try generating our own two part counterpoint following Haydn's cantus firmus, and fixing the first and last notes. To help create better quality counterpoint we add contraints to limit the number of leaps (horizontal intervals of more than a major third) to at most one and require at least six instances of contrary motion, which turns out to be maximal. Note that it would be extremely difficult for a human being with no computer help to generate counterpoint following such restrictions, but the SMT solver instantly returns the solution shown in Figure 3.

3 Selected Features

text here

Acknowledgements

Thanks to Youyou Cong for many enlightening discussions about type theory and music over the past several years, and for suggesting to look at Beethoven's exercises as a source of examples. Emina Torlack's class CSE 507 was invaluable for learning the fundamentals of SMT solvers, and Stephen Rumph's classes MUHST 211 and 430 introduced me to galant schemata.

References

E. Aldwell and A. Cadwallader. 2018. *Harmony and Voice Leading.* Cengage Learning.

Clark W. Barrett, Aaron Stump, and Cesare Tinelli. 2010. The SMT-LIB Standard Version 2.0.

Youyou Cong. 2022. Towards Type-Based Music Composition. https://drive.google.com/file/d/1GZalMD3T5YFVfIO4xeUL4G35Y4pURFDE/view. Presented at TFPIE 2022.

Youyou Cong and John Leo. 2019. Demo: Counterpoint by Construction. In *Proceedings of the 7th ACM SIGPLAN International Workshop on Functional Art, Music, Modeling, and Design* (Berlin, Germany) *(FARM 2019)*. Association for Computing Machinery, New York, NY, USA, 22–24. https://doi.org/10.1145/3331543.3342578

Levent Erkök. 2022. SBV. http://leventerkok.github.io/sbv/.

Johann Joseph Fux. 1965. *The Study of Counterpoint.* W. W. Norton & Company.

Robert Gjerdingen. 2007. *Music in the Galant Style.* Oxford University Press, Cary.

Kent Kennan. 1999. *Counterpoint : based on eighteenth-century practice* (4th ed. ed.). Prentice Hall, Upper Saddle River, NJ.

John Leo. 2022a. Counterpoint. https://github.com/halfaya/Counterpoint.

John Leo. 2022b. Music Tools. https://github.com/halfaya/MusicTools.

Alfred Mann. 1987. *Theory and practice : the great composer as student and teacher* (1st ed. ed.). Norton, New York.

Gustav Nottebohm. 1971. *Beethoven's studien.* M. Sändig, Niederwalluf bei Wiesbaden.

Nadia Polikarpova, Ivan Kuraj, and Armando Solar-Lezama. 2016. Program Synthesis from Polymorphic Refinement Types. *SIGPLAN Not.* 51, 6 (jun 2016), 522–538. https://doi.org/10.1145/2980983.2908093

Julia Ronge (Ed.). 2014. *Beethoven Werke XIII: Kompositionsstudien bei Joseph Haydn, Johann Georg Albrechtsberger und Antonio Salieri.* G. Henle Verlag, München.

Niki Vazou, Eric L. Seidel, Ranjit Jhala, Dimitrios Vytiniotis, and Simon Peyton-Jones. 2014. Refinement Types for Haskell. *SIGPLAN Not.* 49, 9 (aug 2014), 269–282. https://doi.org/10.1145/2692915.2628161