Music Static Analysis

Collier Crisanti
University of Virginia
cc5ny@virginia.edu

Will Leeson
University of Virginia
wel2vw@virginia.edu

ABSTRACT

Static analyses are a variety of techniques that can be used to examine code. These examinations can find potential problems, such as assertion violations or array out of bounds errors. Linters are a form of static analysis tool that can enforce good programming practices and find possible bugs in code. While the issues they find may be legal in the programming language, they may be issues nonetheless. Sheet music can be thought of as a set of instructions to produce music. The musician is the compiler. Along with the musician, the instrument is the execution platform. There are various rules a piece of music should follow. The piece should stay in the key of the music. Instruments should play notes they are physically capable of playing. There are also genre specific rules. With this in mind, we introduce a new tool for statically analysing music. Through the use of a rule set, this tool with examine the music for potentially problematic patterns and report them.

CCS CONCEPTS

• **Software Engineering → Static Analysis**.

KEYWORDS

static analysis, lint, music, counterpoint

ACM Reference Format:
Collier Crisanti and Will Leeson. 2019. Music Static Analysis. In *Proceedings of ACM Conference (CS6501).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Static analyzers are used to check for potential issues in the source code of a program. The types of issues they can identify range based on the type of tool. Symbolic execution tools can find the reachability of sections of code. This allows for tools to find when error statements can be reached. These tools require understanding of the values variables can take and how this will effect execution. As such, they are quite expensive to run. Linters are a more computationally cheaper form of static analysis. They can identify potential patterns in code that may cause issues. These issues can be as detrimental as potential buffer overflow, or as benign as changing the way code is formatted so it is more readable.

In a similar vein to how source code provides the instruments for a program, the score of a musical piece is in a sense the "source code" of that piece. There are various patterns that can be applied to a score. For example, staying in the key of the piece, or playing a note outside of an instruments range. However, most of the patterns are less absolute and more genre specific.

In this paper we seek to examine if and how a static analyzer could be build to perform similar functions as it does on source code, to the scores of music. To do this analysis, we present a rule based tool that is similar to a rule based linter that analyzes a representation of sheet music.

2 POSITION IN THE FIELD

Analyzing music using different software engineering techniques can have various benefits. In the literature we have seen, tools doing this analysis have been targeted at teaching students various concepts in music, like how to write sheet music or how music can be interpreted.

A paper published by Ruiz-Rube et al. looked to create a tool that could be used in general by domain specific languages[7]. In this paper, the authors completed a study that looked at musical students ability to write sheet music. Their approach uses grammars to define their rule violations. The rules that they chose were designed by professional musicians and academics in music. Their rules are primarily about the way the sheet music was written. They check things such as the composer wrote the tempo of the piece, the composer titled the piece, and the composer is listed on the piece. While it is important to list this information, our project differs in scope. Our project aims to look deeper into the music and reason about how the piece will sound when executed. However, verifying these properties are well within the scope of what our tool could do.

Desainte-Catherine et al. present a way to analyze how scores interact[5]. Using their definition of interactions, they allow for different "interpretations" of pieces that the composer can define. Using these "definitions", a performer can alter their performance of a piece. Using inputs from the crowd, they can decide how the performance should proceed using a graph with transitions that are triggered based on this input. They then statically analyze these graphs to find "safe" executions. They define safe as that it follows the temporal constraints placed on the music. Our tool does look at the way instruments interact during a song. However, deciding how a song should be played and deciding if it was played right is dynamic analysis, which our tool does not handle.

In another paper by Desainte-Catherine et al., they present a way to represent timing constraints of a piece using Timed Automaton[4]. These timed automaton can be used to describe certain invariants and properties about a piece. An example they give of these properties would be with song structure. For example, a piece could be defined that whenever a verse occurs, it must be

CS6501, December 2019, Charlottesville, VA, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/1122445.1122456

followed by a chorus. They then verify this property using model checking. Assuming the composer annotated the music with these properties, like what is a chorus, bridge or verse, these structures of song could easily be divined by our tool.

3 MOTIVATION

When composing music there are various patterns that are followed to ensure the best sounding music. Certain chord progressions do not lend themselves to genres of music well, some intervals can be jarring to a listener, etc. While these can be noticed once heard aloud, detecting theses during composition would allow for faster editing. If someone with no musical experience is attempting to begin composing, a tool that accomplishes this detection can guide them away from common pitfalls. An experienced composer who wants to write music for a new genre could find it useful to have a tool that shows what rules they should follow in this new style. To that end we try to present a tool that can show the potential of applying static analysis techniques to music.

4 PROBLEM

The goal of this project was to define a way to statically analyze music. This problem can be separated into two sub-problems. The first problem is how to accomplish the actual analysis. We propose that a set of rules can be written that describe issues that can occur in a mscx representation of sheet music. These rules can deal with individual notes, notes interacting with each other, or an entire piece. These rules can then be checked against the data of the piece and the tool will then report where any violations occur.

The more nebulous problem is writing the actual rule set. This requires knowledge in music theory and whatever type of piece that is being evaluated. Not all rules are general purpose, so they must be written with the pieces they will be applied to in mind. Even with a genre in mind, it may be difficult to define rules. We approach this problem by researching a specific genre with concrete rules, counterpoint. We then implement these rules and create a tool that can evaluate counterpoint pieces.

5 APPROACH

To analyze sheet music, we use the following steps. First, we format the music in a style that can be parsed. This involves converting it into a domain specific XML document called an mscx file. Next, we define a rule set that can be applied on the music. These rules can be general purpose or specific to a genre or style of music. Next, we create a parser that can obtain the meta data we need to analyze the music. This includes the type of instrument and the notes it plays. The parser will produce an intermediate representation we can use to apply the rules. Finally, we apply the rules and report the violations and any information necessary to fix them. Our tool was written in one Python3 file with approximately 400 LOC.

5.1 Formatting Sheet Music

The first step that must be taken is formatting the sheet music that will be analyzed. This can be done using a variety of tools, or can even be done by writing the mscx document. We use the tool MuseScore to generate any mscx document we have analyzed[2]. Ideally, a user of our software will be a composer using one of these

```
rule in_key(prev_note, curr_note, \
            next_note, key):
    if next_note in key:
        return True
    else:
        if is_passing_note(prev_note, \
                           curr_note, \
                           next_note, key):
            return True
        else:
            return False
```

Figure 1: Rule to check if note is in the key of the song

software tools to make their music. Depending on the size of the piece, it is feasible that a user can hand translate a physical piece of sheet music into MuseScore, and then MuseScore can produce the mscx file for them.

5.2 Definition of Rule set

Next, we define our rule set. This is the most extendable portion of this work and also the most difficult to define. As music is subjective, there are few rules of what is "incorrect" in the general sense. There are impossible feats, such as playing the lowest note, highest note, and middle C on a piano. There are notes instruments simply cannot play. These are rules that are well defined. There are also taboos in music. In traditional classical music, the interval of a tritone should be avoided[6]. This interval was deemed unholy, and of the devil. A tritone is an interval of six semitones. For example, B to F and G to C# are both tritones. In most music, it is also generally agreed that a piece should stay in key. The exception would be for passing tones. These are notes that are not in the key of the music but are between two notes that are in the key.

This posed an issue with this project. In order to circumvent this problem, we found a style of music with concrete rules and enacted them. As both of the authors have varying music training, this was not an impossible task. A possible extension of this work would be to collaborate with classically trained musicians, or music theorists to define more rules.

To create a rule for our tool, we first define it as a function in python. This function can leverage the meta-data present in an mscx file or data a user defines to find violations. Refer to Figure 1 as an example of a rule. This rule checks if the note in question is in key. If it is not, it checks if the if the note is a passing note through the function `is_passing_note()`.

In terms of classic static analysis, the closest analog would be the tool PMD [8]. PMD is a Java source code analyzer that can find issues in source code of varying levels of importance. These can range from code breaking issues to problematic code styles. Like PMD, our tool can benefit from extending the rule set.

5.3 Parser

Since the mscx file type is an XML derivative, we were able to use the ElementTree XML API[1] from the Python3 standard library. These

```
for each instrument
    get instrument information
    for each measure
        for each voice
            for each chord and rest
                get duration
                if chord
                    get notes
```

Figure 2: Parser Algorithm

files contain a large section of meta-data at the start of the file, for things such as author and title. In addition it also contains various styling information that is vital for composing programs to be able to properly render all the various information in a score besides notes. For our program we are only concerned with information on that is contained in the staff. To that end, our parser only targets information in the "Staff" tags.

We start by retrieving the information of the instrument corresponding to the track. Once the instrument information has been obtained the parser will start to loop through the measures associated with the staff. Each measure contains voices (or layers); however multiple voices can only exist for digitally played music. For our study of Baroque counterpoint, we do not have to worry about this. If this tool was to be extended for more recently composed music, multiple voices would need to be addressed. Once we are inside our "Voice" tag, we loop through the "Chord" and "Rest" tags. Both contain their own respective duration (quarter, half, etc.), and the former contain one to many "Note"s. Despite not technically being chords, single note are placed under these tags.

Notes have multiple tags, but we care about their associated pitch and tonic pitch class (tpc). The pitch that is linked to a note is not the actual frequency in hertz that the note should be played back in. Instead, it refers to the Musical Instrument Digital Interface (MIDI) tuning. MIDI is a standard for digital music information transferring, and this MIDI pitch along with the tpc allow us to obtain a full representation of a note. The tpc specifics the letter representation of a note, for example an F♯ and a G♭ sound identical but can be written in either form. Both values can be looked up in a table to get their relevant information.

Throughout the process this information is being stored into our defined classes of the music. Refer to Figure 2 for the algorithm of the parser.

5.4 Applying Rules and Reporting Violations

Once we have a representation of the score that our program can interpret, we can start going through and looking for rule violations. For most of the rules, we can go through on a per instrument basis. While going through each instruments track we look at the current note, one note behind the current, and two notes behind the current. For most of the rules, we can feed them this moving window of notes to check for violations.

For two of our rules, we need to examine multiple instruments at once. We can simply have multiple passes on instruments occurring at the same time, but we also have to keep track of when the notes of the instruments line up. To this end, we count the the time value of the notes, and attempt to match up the times as best as possible. For example, if one instrument plays a half note (we would assign this a time value of .5) and another instrument plays two quarter notes (each with a time value of .25), we would check the rules at the first note. After, we would move to the next note on the first instrument (that played a half note), but the other instrument (with two quarter notes) would move forward two notes.

6 STUDY

To test our tool, we looked at comparing pieces by composers of varying fame in the same style. We chose the style counterpoint. This style of music first rose to prominence during the Renaissance and continued to be popular into the Baroque period. We chose this style because it has clearly defined rules. These rules were explained by the composer Johann Joseph Fux in his book *Steps to Parnassus* [6]. We have implemented 8 rules that are common to all forms of counterpoint in this book. Using these rules, we will look for violations among composers of three levels of popular.

We define popularity using the number of pieces a composer has had played at Carnegie Hall. Carnegie Hall is a famous venue for orchestral music that has been open since 1891[3]. We argue that if a composer has had more pieces played at one of the most popular venues in the world than a different composer, then they are more popular than the latter. For our study we define low popularity as having their music performed less than one hundred times, medium popularity between one hundred and two hundred, and high as more than two hundred times. Since they have recorded which pieces have been played there since their opening, we have over 100 years of performances to form our metric.

The hypothesis of this study is that pieces from more popular composers will have less violations than pieces from less popular composers. If the rules of counterpoint define what "good" counterpoint is, then pieces that follow these rules should be better pieces. Better pieces should be more popular, and popular pieces should come from more popular composers.

Our tool and the data used in this study is located at https://github.com/CCrisanti/music_analyzer/.

6.1 Rule Set

In this section we define the rules for this study.

Rule 1: Avoid tritones over three note intervals. As discussed earlier, tritones were disliked in classical music, and this idea is preserved in counterpoint. This rule was implemented by looking at three notes and checking if they spanned 6 semitones without the middle note going outside this interval.

Rule 2: Check if interval is of correct type. In counterpoint, certain intervals between notes are allowed. These include minor and major seconds, minor and major thirds, perfect fourths, perfect fifths, octaves, and minor sixths. In the case of minor sixths, they must be ascending minor sixth and they are followed by downward motion. This means that the note is eight semitones above the previous note and the following note is lower than it in pitch. This rule is implemented by making sure the difference in pitch between notes

Table 1: Average Violations of Rules by Composer Popularity

Rule	Low Popularity Composers' Violations	Medium Popularity Composers' Violations	High Popularity Composers' Violations
Avoid Tritones	1.974	1.364	13.120
Use Only Allowed Intervals	21.333	7.091	85.040
Multiple Skips In Opposite Directions	21.077	10.227	80.960
Skips in Same Direction form Consonance	17.615	8.591	59.400
Begin and End on Perfect Consonance	0.000	0.000	0.000
Contrary Motion is the Norm	2.375	2.758	2.364
Approach Perfect Consonance	102.821	153.181	153.04
Avoid Interval of Tenth	0.026	0.000	0.00
Total	**167.385**	**177.212**	**393.924**

is one of the accepted intervals. If the interval is of a ascending minor sixth, we check that it is followed by downward motion.

Rule 3: Consecutive skips should be in opposite directions. A skip is when two notes are more than two semitones apart. In counterpoint, incremental changes are the norm. If there is a large jump, the song should attempt to return by counteracting a skip. This rule is implemented by checking if the differences between a note and its predecessor and the predecessor and its predecessor are more than two semitones. If they are, then we check if they are in the opposite direction.

Rule 4: Consecutive skips in the same direction should form a triad. The second skip should be smaller than the second. This rule follows the previous rule to give the composer a way to have consecutive skips in the same direction. They prevent large skips by saying the skips must form a triad. A triad spans seven semitones. The first and last notes of a triad will also form a consonance since they form a perfect fifth. This rule was implemented by checking if the Rule 3 was violated. If it was, then it checks if the first skip was bigger than the second. Then it checks that the interval forms a perfect fifth.

Rule 5: Counterpoint should begin and end on perfect consonances. Counterpoint deals with multiple "voices" interacting, where a voice is an instrument or person's voice. Each piece should begin and end with a perfect consonance being formed by at least two voices. An exception would be if only one voice is playing at the start or end of a piece. This rule is implemented by checking if for all voices against all other voices the first and last notes form a perfect consonance. Perfect consonances are defined as the same note, a perfect fifth, a perfect fourth, or one octave different[9].

Rule 6: Contrary motion should dominate a piece. Contrary motion means that two voices should move in opposite directions. This means if one voice raises in pitch, another should lower in pitch. Counterpoint deals with voices playing against each other, and one way they achieve this is through contrary motion. This rule is implemented by checking for each instrument against each instrument, at least one should rise in pitch while the other lowers in pitch. If all but on instrument change in one direction, but one goes in the other direction, this rule is not violated. However, there will still be n choose 2 violations of this rule. To handle the false positive, we normalize the results by dividing them by n choose 2.

Rule 7: Perfect consonances must be approached through contrary or oblique motion. While the previous rule suggested contrary motion be the norm, this rule requires that either contrary motion or oblique motion is needed when perfect consonances are approached. This suggests that this rule is more important as it is a requirement instead of a suggestion. Oblique motion is when one voice raises or lowers in pitch while the other stays the same. This rule is implemented similar to Rule 6, except that it also allows for oblique motion. It also checks if the motion is followed by a consonance.

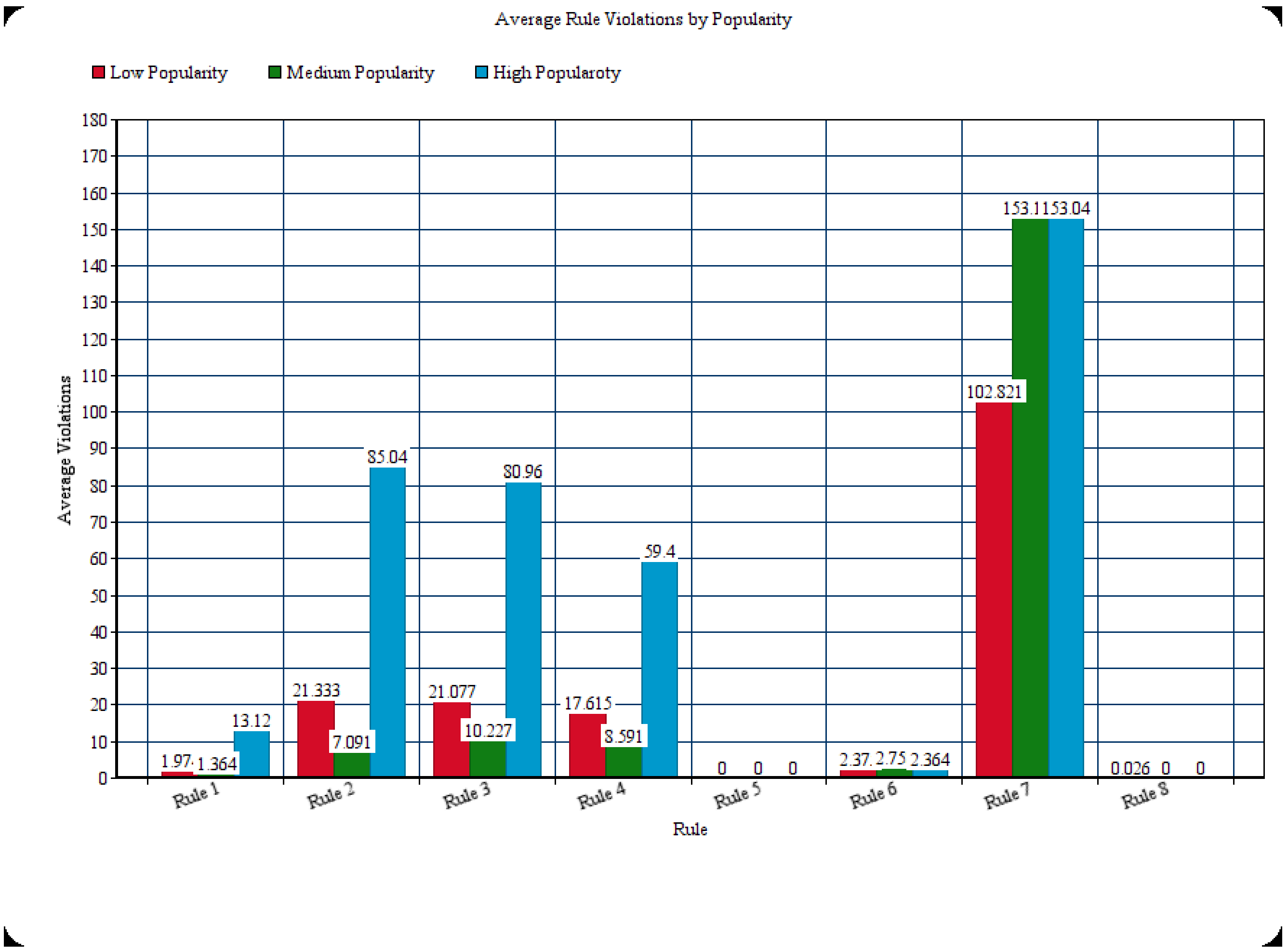
Rule 8: Interval of a tenth should be avoided. As mentioned earlier, counterpoint generally has incremental changes. An interval of a tenth is a compound interval made of two accepted intervals. It spans an entire octave and then an additional third. Thus, this is generally avoided because of how large a change it is. This interval is also explicitly listed as poor for counterpoint. This rule is implemented by checking if two notes span 15 or 16 semitones.

6.2 Data

For our study we examined eighty-seven (87) songs, from fifteen (15) different composers, ranging in popularity from Johann Sebastian Bach to Vincenzo Galilei (not to be confused with his son). For each composer we looked at approximately five of their pieces, except for Johann Joseph Fux, who had twenty-one (21) pieces. We also kept the pieces to under 5 pages to try and curb any skewed results. The pieces ranged from three to seven different instrumental tracks, with our most common instrument being the piano.

6.3 Results

The results of our study may be found in Table 1. For most rules, Medium popularity composers had the lowest number of violations and high popularity composers had the most violations. The one rule low popularity composer did best on average was Rule 7: approaching perfect consonances through contrary or oblique motion. The only rule high popularity composers did the best in was Rule 6: contrary motion is the norm. However, they only did minutely better. Beginning and ending on a perfect consonance was never violated. Overall, Low popularity composers did the best, but if we ignore rule 7, as this rule greatly inflated scores, medium popularity composers had the least violations on average.



6.4 Discussion

For the purposes of this discussion, we ignore rule 7. This rule had far more violations than any other rule and inflated our violation counts.

Based on our results, it is clear our hypothesis was wrong. High popularity composers broke all but three rules more than the other two groups. On average, they had twice as many violations, even if we ignore rule 7. The pieces written by the composers are still highly regarded. A reason for their rule breaking is most likely not ignorance, but instead due to their knowledge. These composers are considered masters of their art forms. This means, they likely knew the bounds of these rules. When they do break a rule, it may be because they know it will still sound correct. Low popularity composers had the second most violations, ignoring rule 7. This may be attributed to their ignorance or lack of mastery of the technique. Finally, medium popularity composers had the fewest violations. This may come from a better understanding of the paradigm. However, they may not have been skilled enough to break the mold and make new, interesting pieces when it fit.

7 CONCLUSION

In this paper, we present a tool to statically analyze music. This is an important problem as it may help produce better composers and help other composers branch out into new fields. This branching out may help diversify currently stagnant fields. Statically analyzing music has it challenges. The one most clear to us is defining an oracle: what is "correct" music. As music is subjective, it is important not to stifle creativity, yet there are still agreed upon truths and taboo practices. An extension of this work would be to work in tandem with trained musicians and music theorists to write more rules.

We also present a study analyzing pieces of composers of various levels of popularity. While our hypothesis was incorrect, we still find that this study was successful in showing how one could statically analyze music. We were able to parse sheet music and find locations that disagreed with the rule set that was laid out by classically trained musicians. These lays the ground work for different rule sets made specifically for a genre in question.

ACKNOWLEDGMENTS

To Mitchell Gerrard, who's knowledge of Bach and classical music helped us target our study

REFERENCES

[1] [n.d.]. 20.5. xml.etree.ElementTree - The ElementTree XML API¶. https://docs.python.org/3.6/library/xml.etree.elementtree.html
[2] [n.d.]. Create, play and print beautiful sheet music. https://musescore.org/en
[3] [n.d.]. Performance History Search. https://www.carnegiehall.org/
[4] Jaime Arias, Jean-Michaël Celerier, and Myriam Desainte-Catherine. 2017. Authoring and Automatic Verification of Interactive Multimedia Scores. *Journal of New Music Research* 46, 1 (2017), 15–33. https://doi.org/10.1080/09298215.2016.1248444 arXiv:https://doi.org/10.1080/09298215.2016.1248444
[5] Myriam Desainte-Catherine and Antoine Allombert. 2005. Interactive scores: A model for specifying temporal relations between interactive and static events. *Journal of New Music Research* 34, 4 (2005), 361–374. https://doi.org/10.1080/09298210600578212 arXiv:https://doi.org/10.1080/09298210600578212
[6] Johann Joseph Fux and Alfred Mann. 1971. *The study of counterpoint: from Johann Joseph Fuxs Gradus ad parnassum.* W.W. Norton.
[7] Javier Merchán Sánchez-Jara, Tatiana Person, Almudena Mangas-Vega, Iván Ruiz-Rube, Juan Manuel Dodero, and José Mota Macias. 2018. Static code analysis to measure the quality of musical scores.
[8] Pmd. [n.d.]. PMD. https://pmd.github.io/
[9] Michiel Schuijer. 2009. *Analyzing atonal music: pitch-class set theory and its contexts.* University of Rochester Press.