## Abstract

Music theory is the discipline of composing and studying music, often framed as systems and rules, such as scales, chords, and rhythms, as well as higher-level aspects like song structure.

This paper and accompanying program explore the relationships between music composition and music theory rudiments and computer programming using Scheme, a functional list processing language. The program relies on the strengths of the language, and interprets musical structures using functions on finite state machines implemented as linked lists. An intuitive interface for building musical components such as scales, chords, and measures is defined using functions that operate on sets of musical notes and/or time. These components are then used to build musical pieces, utilizing some degrees of randomness for diversity.

This research has found that functional languages in the list processing language family can build a powerful and intuitive interface for music composition and music theory. It is hoped that this work will lead to more study in the area of generating music using computers or technology, as well as extensions onto this implementation to expand the rules enforced when generating the songs to further create interesting music.

## Introduction

This paper discusses music composition programming using Scheme and creating and modifying musical elements using pure functions, as well as defining functions to produce human-readable music.

In Scheme, much work has been done in the realm of programming to directly produce audio using midi programming. However, this requires manually defining the desired output using Scheme syntax and does not provide a method for algorithmically creating human-readable music.

## Background

### Music Composition

#### Terms

Scale degree: A scale degree is a possibly numeric representation of a particular note on a scale relative to the root note. [1] For example, in the scale of C major, C has scale degree 1, and G has scale degree 5.

Scale: A scale is a set of musical notes, usually built by following a root note and a set of scale degrees.

Chord: Like scales, a chord is a set of notes. Notes in a chord, however, are meant to be heard simultaneously. Broken chords, notes of a chords not played simultaneously, can also be considered chords.

Time signature: The time signature describes how many beats are in a measure, and which note value is the beat is equivalent to.

Note:

Measure

### Lilypond

Lilypond is a GNU Project software suite used for composing music. Unlike other popular composition programs, Lilypond is intended to be used with a text editor and defines a descriptive syntax for sheet music, which is compiled to PDF or MIDI. This concise syntax lends itself well to automation by other programs. Lilypond is written using C++, Python, and Scheme.

### Functional Programming

Imperative programs are structured as a series of instructions which change the state of the program, with a guaranteed instruction order. In the functional program paradigm, programs are built via function declaration; functions are first-class citizens and can be passed as arguments to other functions as well as returned from functions. Functional programs, therefore, tend to be constructed by declaring, combining, and composing functions, and can be easier to debug and verify correctness by minimizing state and side effects.

Pure functions are functions that is treated as a mathematical function with deterministic output, i.e. the output of the function is determined only by the input, and using the function does not change the state of the system or introduce side effects. Pure functions were used when programming the music programming system in order to create a more modular system open to extension and future work.

### Scheme

Scheme is a functional language of the Lisp family. Scheme’s primary data structure, the singly linked list, is represented as pairs of parentheses with elements contained within. The syntax of Scheme is likewise represented using linked lists of function applications and parameters using prefix notation. This unique marriage between code and data can be leveraged to construct declarative systems, such as LAML, which uses Scheme as a markup language, meaning documents can be evaluated as a Scheme program.

This research uses the Simply Scheme implementation of the language in order to highlight how a simple language specification can produce powerful high-level constructs like musical elements.

### Linked Lists

## Goals

The goals of this project are to:

* Create an intuitive interface for building musical structures in a LISP-family language
* Show uses of linear linked lists for representing abstract data
* Provide an easily extendable suite for creating and implementing algorithms for music composition

## Music Composition in Scheme

### Scales and Chords

Musical scales are a set of notes considered to be part of the scale, defined by a set of scale degrees and a root note to. In Western music, all scales, when considering the sets of notes represented by those scales, can be interpreted as subsets of the chromatic scale, where the chromatic scale is defined as the set of all possible notes.

These interpretations lead to a very intuitive implementation in Scheme using the higher-order *filter* function. The chromatic scale is defined as a constant list of all possible notes. Enharmonic notes, notes that are equivalent to another note musically but written differently, are excluded from the list for the sake of simplicity. To apply the *filter* function to the chromatic scale, a predicate function is needed, so lists of valid scale degrees are declared (note that in music theory, the degree from a note to itself is 1, not 0), and the predicate determines whether the degree from the root to a given note is a valid degree:

(define (filter-notes root scale degrees)

(filter (lambda (to)

(list? (member (scale-degree root to scale) degrees)))

scale))

Using this implementation is intuitive:

(scale “c” major)

This definition is applicable to other musical structures defined by degrees between a root note and other notes, as in the case of chords. Chords can be interpreted as a subset of a scale, much like how scales can be interpreted as subsets of the chromatic scale:

(define (chord root scale degrees)

(reorder-for-root

(filter-notes root scale degrees)

root))

Describing chords functions much like describing scales:

(chord “c” (scale “g” major) triad)

An implementation of building simple chord progressions is also provided, including a predefined I-V-VI-IV progression, the most commonly used progression in Western music. The chord progression function takes a set of degrees and constructs an ordered list of chords at those degrees for the given scales, playing to the strengths of the Scheme language.

Move to further research The supplied implementations for chords and scales reorder the notes, such that the root is always the head and the notes are sorted from lowest to highest degree. This is only for convenience of the implementation, and different orderings of notes might be needed by the composer, to create certain modes or chord inversions for example. In that case, an ordered set of degrees or list indices could be defined to create reordering functions.

### Time Signatures, Notes and Measures

Time signatures, notes, and measures are the simplest structures in the program, and the function definitions are given to aid in code readability and to create a more descriptive syntax. Time signatures are represented as a list of (number of beats in measure, value of whole note), much like in sheet music:

(define (time-signature num-of-beats note-value)

(list num-of-beats note-value))

Notes are likewise defined as a list of (pitch, length) as so:

(define (note pitch length)

(list pitch length))

Finally, measures are defined as a consecutive list of notes:

(define (measure notes)

(list notes))

Scheme’s higher order functions make validating musical measures simple. To find the length of a measure, all note lengths in the measure need to be collected and summed. Scheme’s higher order *map* function transforms the list of notes in the measures to a list of only the note lengths, and the *apply* function applies a function to a list as if the given list were parameters to that function:

(define (measure-length measure)

(apply + (map cadr measure)))

### Algorithms for Composing Music

This program defines a song with the following format:

(key-signature time-signature (measures))

An implementation for composing music is provided, using a simplified version of the waveform collapse algorithm to validate note and measure lengths. The possible states of the algorithm are represented as a set of note patterns is given to the algorithm, with each note pattern potentially having different lengths. In this case, the only rule is that the length of the measure must be equal to the length given in the time signature. For each measure, random note patterns are chosen until the measure length is either valid, in which it will continue to the next measure, or the measure is too long, in which there is a contradiction and the measure generation is repeated from the beginning. Chord progressions are used in this function to add some musical complexity; the song begins at the root chord and iterate through the progression each measure, repeating when necessary, and notes are chosen randomly from the chord corresponding to the measure.

### Output

The Lilypond program defines a descriptive syntax for describing music that it uses to generate PDF files of sheet music. This program provides a Scheme module that generates valid Lilypond code for songs and musical elements following the correct definitions.

## Conclusion

## References

1. <http://www.dbnet.ece.ntua.gr/~adamo/languages/books/p359-hudak.pdf>
2. <https://people.eecs.berkeley.edu/~bh/ss-toc2.html>
3. http://people.cs.aau.dk/~normark/laml-distributions/laml/index.html
4. Kolb, Tom (2005). *Music Theory for Guitarists*
5. <https://dl.acm.org/doi/pdf/10.1145/1890703.1890704>
6. https://lilypond.org/introduction.html