## 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

## Background

### Music Composition

#### Terms

Interval

Scale

Chord

Time Signature

Note

Measure

### Lilypond

### Functional Programming

### Scheme

### 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 intervals and a root note to begin the intervals from. 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, a function that operates on a list and builds a new list consisting of elements of the original where the given *filter* function applied to those elements evaluated to true. 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 intervals are declared (note that in music theory, the interval from a note to itself is 1, not 0), and the predicate determines whether the interval from the root to a given note is a valid interval:

(define (filter-notes root scale intervals)

(filter (lambda (to)

(list? (member (interval root to scale) intervals)))

scale))

Using this implementation is intuitive:

(scale “c” major)

This definition is applicable to other musical structures defined by intervals 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 intervals)

(reorder-for-root

(filter-notes root scale intervals)

root))

Describing chords functions much like describing scales:

(chord “c” (scale “g” major) triad)

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 interval. 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 intervals 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

## Output