MIDILC: A MIDI Language Compiler for Programmatic Music Composition

Alex "Akiva" Bamberger
Benjamin Mann
Frederic Lowenthal
Ye Liu

Contents

Chapter 1

Introduction

The language, hereafter referred to as MIDILC (pronounced MIDDLE C, standing for MIDI Language Compiler), allows programmers to compose music. It compiles into MIDI format and has syntax that is similar to Java, changing the basic primitives and the meaning of various operators (Fig 1.1).

The langauge is dynamic, but introduces types for the programmer most comfortable with the syntax of C or Java. Types must be used upon variable declaration, but are left optional for function declarations and arguments. Each of the above primitives supports the + operator, which increments the duration of each note and the .+ operator, which increases the pitch of each note. Each type can be safely cast up in the following order: Number Note Chord Sequence. The standard library, written in the language itself, supports major and minor chords, arpeggios, repetition, and other such basic and often used concepts. Note durations are specified in terms of whole notes (w), halves (h), quarters (q), eigths (e), and sixteenths (s). Sequences can either be appended to, which advances the "current time" by however long the appended sequence is, or else something can be inserted into a sequence at a given offset using bracket notation. Functions are specified in the same way as in C.

Sequences and Chords can simply be outputed to the intermediate representation (IR) of CSV format using the play() function. In order to actually write the MIDI files, the CSV is fed to a Java program that interprets the CSV using the javax.sound.MIDI package.

Composing music on a computer is often done using GUIs that allow the user to drag and drop notes or using instrument inputs. This lets the musician hear his compositions as he is creating them, and often gives the musician a simple MP3 or MIDI ouput. As computer scientists, the MIDILC team finds such methods tedious, extraneuous and requiring too much natural music talent. MIDILC follows the virtues of any great programmer: laziness,

Number	a value between -2^{31} and $2^{31}-1$
Note	a musical note with pitch and duration
Chord	stores notes with equal durations and start times, represented by an integer list
Sequence	a sequence of notes and chords, represented by a list of integer lists

Figure 1.1: Types in the MIDILC language.

hubris, and impatience [1] and propose a language for those wishing to turn their quantitative skills into beautiful music.

MIDILC also attempted to redress other issues of regular music composition. Songs often have frequent recurring themes. Manually reusing these themes requires precision and dedication. If pieces of a song could be manipulated automatically for reuse and slight modification, song production speed could increase dramatically. The MIDILC language allows programmers to algorithmically generate notes, chords, and sequences of notes and chords by writing functions. This allows for writing interesting compositions that minimize time spent rewriting basic MIDI manipulation routines and implementing primitive musical constructs. The language works optimally with compositions that make consistent use of simple motifs as it encourages reuse of simple mathematical operations on notes and chords.

MIDILC is tailored for crafting melody sequences. For a sequence containing a series of whole notes, one could easily manipulate the notes in the melody to create sequences of counterpoints to the melody. Using MIDILC, a Major() and Minor() function are easy to craft and chords simple to arpeggiate to make counterpoint. MIDILC allows for the simple concatenation of sequences, and so complicated sequences could be easily made from simple starting blocks.

[1] Wall, Larry. *Programming Perl*, O'Reilly 2000.

Chapter 2

Language Tutorial

Chapter 3

Language Reference Manual

Chapter 4

Project Plan

Chapter 5

Architecture & Design

Chapter 6

Test Plan

Chapter 7

Lessons Learned

Chapter 8

Appendix