G♭: Language Specification

Ezra Joffe-Hancock and Zach Stein-Perlman

Spring 2022

1 Introduction

Ezra plays alto sax in a jazz combo here, and when he's struggling with soloing over certain songs, it helps him to run over the chord changes and play along with the recording, but the best is when there's a backing track on YouTube, so he can play along with the chords without drowning whoever is soloing on the recording. However, there are often not backing tracks for songs, or only backing tracks that are played in certain keys, so there's a need for more flexibility and availability of backing tracks. There are music notation software systems that allow you to write songs and chords, but these often take a while to write, as they're designed for a greater range of features. In other words, they're too flexible and open-ended for the purposes of a quick backing track.

We created a language that makes it easy to quickly create a backing track to play over. The programmer creates sections of music by arranging chords and choosing their durations, then describes how many times and in what order to play each section. The output is an XML file that describes the corresponding audio. MusicXML is a standard sheet music file representation that can be interpreted by many different third-party programs, and contains information that allows the music to be both visually displayed and produce audio output (as it is MIDI compatible). We leave the decision of what program to use to interpret the XML file up to the user, as there is a large range available, but note that SoundSlice (`https://www.soundslice.com/`) is an online free program with a neat user interface that does so.

2 Design Principles

The language is simple and accessible for a non-technical user, and it should be easy to quickly translate an existing written chord progression into a program. The syntax is natural. The order of commands makes programs easily readable with intuitive layout. Simplicity in implementation is prioritized (worse is better!). A valid program is ideally easy to read and facilitates quickly understanding the music that it describes.

3 Example Programs

Examples can be run by calling

```
dotnet run <input file>
```

on the command line from inside the Gflat directory inside the lang directory. An XML file matching the name of the example is created in the Gflat directory, which can be put into various programs (including soundSlice) to produce musical notation or sound output. Below we see example Gflat programs highlighting different language features.

Example 1: simplest possible

```
let Song = {
Cmaj 1
}
play Song 1
```

Example 2: variety of chords and repeated chorus

```
let Chorus = {
Emaj 2
G#min 2
Bdim 2
Eaug 4
Db7 2
Cmaj7 2
F#min7 2
}
play Chorus 2
```

Example 3: multiple sections defined, sections reused in play

```
let Intro = {
Cmaj 2
}

let Chorus = {
Cmaj 1
G7 1
}

play Intro 1 Chorus 10 Intro 1
```

Example 4: sections used within other sections, number of times they repeat specified

```
let Intro = {
Cmaj 2
}

let Phrase = {
D7 1
A7 2
Fmin7 3
}

let Chorus = {
Phrase 1
Cmaj 1
G7 1
Phrase 2
}

play Intro 1 Chorus 10 Intro 1
```

4 Language Concepts

In order to program in the language, a user should have a basic understanding of chords (so they can specify them by writing out their letters, and understanding different chord qualities) and an understanding of a beats as a way to think about the duration of chords. They should understand also how to transfer information from a written piece of sheet music with chords to information within the language, by copying down chords and putting them in different sections as appropriate to efficiently capture the structure of a song.

5 Syntax

Programs have two parts: defining sections and playing them. When defining a section, we give the name of the section, and its components: chord-duration pairs and references to other sections and how many times to repeat them. A chord is made up of a note (representing the starting note of the chord) and a chord quality (either Major, Minor, Diminished, Augmented, Dominant 7th, Major 7th, or Minor 7th). A chord is followed by an integer specifying the number of beats we want the chord to be played for. In the play section of the program, we refer to an order of section names and the number of times we want each one to be played. If a section is defined multiple times within a program (i.e. a section with the same name is assigned more than once) only the latest definition will be used (thus the practice should be avoided).

```
<expr>       ::= <assignment>+ <play>
<assignment> ::= let ␣ <variable> <ws> = <section>
<section>    ::= <ws> { noise+ } <ws>
<noise>      ::= <ws> <subsection> <ws> | <ws> <sound> <ws>
<subsection> ::= <ws> <variable> <ws> <int> <ws>
<sound>      ::= <chord> <int>
<chord>      ::= <note> <ws> <quality> <ws>
<ws>         ::= ␣*
<note>       ::= A | A# | Bb | B | C | C# | Db | D | D# | Eb | E | F | F#
                 | Gb | G | G# | Ab
<quality>    ::= maj | min | dim | aug | 7 | maj7 | min7
<variable>   ::= <string>
<play>       ::= play content+
<content>    ::= <ws> <variable> <ws> <int> <ws>
```

Note that

```
<variable>
```

may not be “play” or “let”. Encoding this in BNF would be long and unenlightening, so we just mention it here.

6 Semantics

Primitives in G♭ include notes, which are every type of chromatic note with equivalent notes represented both as flats and sharps (e.g. G# and A♭) but not when there is no proper flat or sharp version of the note (e.g. E# and F♭). Chord qualities are a primitive. Section names are another primitive, and are of string. Another primitive are integers, which are used specify the duration of chords, the number of times a section repeats when referred to in another section, and the number of times a section repeats when referred to in play.

Syntax	Abstract syntax	Type	Prec./Assoc.	Meaning
C	Note	Note	n/a	A note
Cmaj	Quality of Note	Chord	n/a	A chord to be played where C stands for the note and maj for the quality
Cmaj 1	Sound of Chord * int	Expr	n/a	A chord and its duration
Chorus	Variable of string	Expr	n/a	The name of a section
Chorus 1	Subsection of Expr * int	Expr	n/a	A section and how many times to repeat it.
sound1 ... soundn	Section of Expr list	Expr	n/a	A sequence of sounds and subsections
let var = {section}	Assignment of Expr * Expr	Expr	n/a	A pairing of a section with a name
play var1 x1 ... varn xn	Play of (Expr * int) list	Expr	n/a	A specification of sections to play
<Assignment list> <Play>	Program of Expr list * Expr	Expr	n/a	A complete program

7 Remaining Work

Potential extensions:

- Implement a program correctness checker; e.g., allowing for inconsistencies in capitalization
- Add instruments beyond piano
- Allow more flexible control over tempo, volume, and octave
- Allow for individual notes and custom chords (where the user specifies all the notes in them if they are not a standard chord)
- Chord names printed above chords in musical notation representation