COMS W4115 Project Proposal: ChordZ

Rafi Hasib

rah2175@columbia.edu

Due Date: June 6, 2012

Resubmitted: June 18, 2012

Introduction

Harmonizing offers a creative way to enhance songs, producing chords that align with the melody. Many musicians either use their ear or a musical score to identify appropriate notes. However, this requires considerable skill, memory, and some working knowledge of music theory. Alternatively, one can use music notation software, but the tools that offer this intelligent functionality require significant RAM, configuration by the user, and repeated point-and-click entry to generate a simple output.

Language Overview

ChordZ simplifies the process by allowing the user to type notes of a melody in an intuitive language. Based on the context of the notes and additional criteria provided by the user, it then determines an appropriate harmony for the melody.

Output will first be specified with in an equivalent textual format, similar to the input. Once converted to LaTeX, it can utilize the MusiXTeX suite to visually represent the melody and harmony within a musical score.

Sample Tokens/Operators

- Note names: `a,b,c,d,e,f,g,r` ('`r`' designates a rest)
- Octave: `1,2,3,4,5,ε` (voices do not typically go above or below these limits)
- Note value: `1,2,4,8,...,ε` (i.e. whole note, half note, quarter note, eighth note)
- Accidentals: `@,#,ε` (flat, sharp, none)
- Chord types: `maj,min,7,maj7,min7,ε`
- Roman Numerals: `I,II,III,IV,V,VII,VII,i,ii,iii,iv,v,vi,vii`
- Group delimiters `[],(),|` (beaming, tying/slurring, barline)
- Function Calling: `{}`

Note Representation

In general, a note will be represented as: `<value><name><accidental><octave>`, using whitespace as delimiters. For simplicity, if the expression does not include a value or operator, it will carry the previous note value and octave. The collection of notes will be stored in a string array.

Chord representation

Chords (either collected in parallel or calculated following note entry) will be stored in three parallel string arrays. The user can provide chords using chord names or Roman numerals, but the compiler will create the array of chord member lists to determine notes for the harmony.

- Array of chord names (e.g. Cmaj, Dmin)
- Array of chord members (e.g. {c,e,g}, {d,f,a})
- Array of Roman numerals (e.g. I, ii – with respect to C major)

Algorithms

Depending on addition information specified by the user (e.g. chord names, key, Roman numerals), the compiler determines certain essential data before calculating harmonies that align with the melody. The following descriptions and pseudocode outline the different type of harmonization implemented by the compiler.

Chord Application

If the user does not specify chords for each note, the compiler uses the notes and key to attribute chord names[1] and Roman numerals[2].

```
get(key,chords)         /* key determines valid chords */
apply(chords)           /* favors tonic, (sub)dominants */
```

Standard Harmony

The user indicates the desired number of harmonizing voices, which affects how the compiler generates voice parts. Where possible, the first harmonization will favor thirds and sixths with respect to the melody. The inclusion of a second harmony will fill the chord to produce complete triads.

```
get(key,chords)
calculate(tenor)        /* favor thirds and sixths */
if hvoices = 2
    fill(voice)         /* favor fifths over doubled root */
```

Four-Part Harmony

Given the melodic line and either (1) chords, or (2) key and Roman numerals, the compiler attributes chords in a variant of *classical* and *barbershop* styles for soprano (melody), alto,

[1] Chord names (e.g. Cmaj, Emin) are absolute names, independent of the key.

[2] Roman numerals (e.g. I, iii) are relative names, dependent on the key. Certain chords within the key are more likely occur than others, allowing the compiler to probabilistically attribute chords.

tenor/baritone, and bass, following 18th century style conventions of complete chords, contrary motion, and avoidance of voice doubling.

```
get(key,chords)
calculate(bass)         /* favor roots and fifths */
fill(voice)             /* fill remainder of chord */
```

Example

Input

The following input represents an excerpt of the tune "Twinkle, Twinkle, Little Star".

```
key{cmaj} 4c4 c g g | a a g r | f f e e d d 2c
```

Output

The output shown satisfies the basic, single-voice standard harmony under minimal conditions. Without any additional information (such as key signature or chords), the program uses the notes provided to determine a key (C major) before approximating chords (e.g. C major, F major, G major). Based on the chords, it finds members of prevalent chords that generate a musical line and produces the voice on its own staff.

```
4e4 e e e | f f e r | d d c c b3 b g |
```