ChordZ Language Reference Manual

Rafi Hasib
rah2175@columbia.edu
COMS W4115: Programming Languages and Translators
Columbia University, Summer 2012

Introduction

The ChordZ language offers musicians a simple programming language for generating harmonies and displaying the content in a meaningful musical score. Using basic melodic input and additional parameters to describe the desired harmony, the built-in algorithms allow the user to realize more complex musical ideas.

The output of the compiler generates plain text for PMX, a pre-processor that handles details in formatting, beaming, slopes, and similar behavior. Its output then produces a TeX file, which MusiXTeX, a suite of music engraving macros, uses to generate the musical score.

Lexical Conventions

Programs use the Unicode character set. Once translated, the sequence of elements reduces to a tokens, comments, and white space. Tokens, separated by whitespace, include identifiers, keywords, constants, operators, and separators. The compiler ignores both comments and whitespace.

Comments

Comments begin with `%` and terminate with a line feed.

Identifiers

Identifiers consist of a sequence of letters, digits, and underscores.

Operators

The language uses equals (=) as an assignment operator. It also uses plus '+' to concatenate two collections of notes.

Keywords

The language has keywords `key`, `harmony`, `song`, and `time` used to describe the music.

Constants/Literals

Notes

The user constructs a `Note` using symbols for the duration, note name, accidentals (if any), duration, and octave, expressed as: `[0-9]?[A-G R][b#]?[0-9]?.`

The leading digit, following the note name, indicates the basic time value: `0, 2, 4, 8,` or `1` (for whole, half, quarter, eighth or sixteenth). The letters `A, B, C, D, E, F` and `G` names the note (pitched at the letter indicated), whereas an `R` designates a rest (a "silent" note). Accidentals are limited to `b` and `#` for flattening or sharpening the note specified. The second digit indicates the octave. In the case the user omits a digit, the note inherits the previous note's values.

Some valid note literals are: `A4, 4Bb5, 8D#, F,` and `3R`.

Chords

Chords can be input in multiple ways, all of which require angled brackets.

- The first way uses the `z` function to take string of note names separated by commas that spell the chord, e.g. `<z(C,E,G,Bb)>` or `<z(D,F#,A)>`.
- The second way is symbolically using `<[A-G][M|m]?7?>`, which the translator decomposes into its individual notes.
- The third approach uses Roman numerals, which rely on the key to place the chord appropriately. Capitalized Roman numerals specify major chords rooted on the scale degree (e.g. `<I>, <IV>, <V>`), and lowercase Roman numerals specify minor chords rooted on the scale degree (e.g. `<ii>, <iii>, <vi>`).

Harmony

The user specifies harmonies relative to the melody as arguments to the `h()` function. Harmonies are specified by integers ranging from -3 to 3, indicating a harmony at most three chord members below the melody to a harmony at most three chord members above. For example, if requesting a (tenor) harmony one above, use `h(1)`.

Punctuation

- The semicolon (;) terminates an expression.
- The vertical bar(|), denotes the end of a measure.
- An '`s`' indicates that the note preceding and following it should be tied.
- Angled brackets, '<' and '>' denote a chord. A chord is assumed to be in effect until it reaches the next pair of angled brackets.
- Curly braces, '{' and '}', denote the region of the melody to harmonize. The user can designate specific portions of the melody to add harmony, rather than the entire region. The braces nest and determine the number of harmonies to include. Without braces, the program simply prints the melody.
- Parentheses surround the arguments passed to a function. Commas separate the arguments passed to a function.

Built-in Functions

a(chord, int)
Generates a simple arpeggiated accompaniment that alternates notes on the resolution of the `int` note value (which can only take on values available for note duration, i.e. 1, 2, 4, 8).

h(song, key, notes, harmony[, harmony, harmony])
Specifies the placement of the harmony with respect the melody. The order matters in instances when fewer harmonies are requested for an excerpt of the music. The function can take one to three arguments (i.e. four voices total). For example, if the melody were a soprano line, the call to request the tenor, alto, and bass harmonies would be `h(-1,-2,-3)`. This function generates a majority of the output passed on to the PMX and MusiXTeX.

k(song)
When the user does not specify the key, the compiler estimates the key based on the available notes. The user can also invoke the function manually at the start using the command above. It analyzes the melody's notes and returns a major or minor chord that it estimates as the tonic chord.

t(int, int)
This function takes two integers as its arguments to display the time signature on the final score.

z(note, note, note[, note])
Builds a chord based on the notes included in the argument). The chord applies until the placement the compiler sees the next chord or barline. For example, `z(C,E,G)` or `z(Bb,D,F,Ab)`.

Sample program

The following program describes the verse of "Hey Jude" and requests harmony for certain areas. During the first harmonization, it only applies one harmony above. For the last three measures it applies both harmonies.

```
key key1 = <F>;
time t1 = t(4,4);

song v1pt1 = 4C4 | <F> 2A3 s 8A 8C4 D | <C> 2G3 4R 8G A |
<C7> 4Bb 4F4 s 8F F E C |;
song v1pt2 = <F> D 1C Bb3 2A 8R {C4 |
<Bb> D 4D 8D 1G 8F 1E s E F 8D | z(F) 2C {8F3 G A D4 s |
<C7> D C R C Bb3 A E F s | <F> F F s 2F}} 4R |;

song verse1 = v1pt1 + v1pt2;
harmony hverse1 = h(verse1, key1, 1,-1);
```