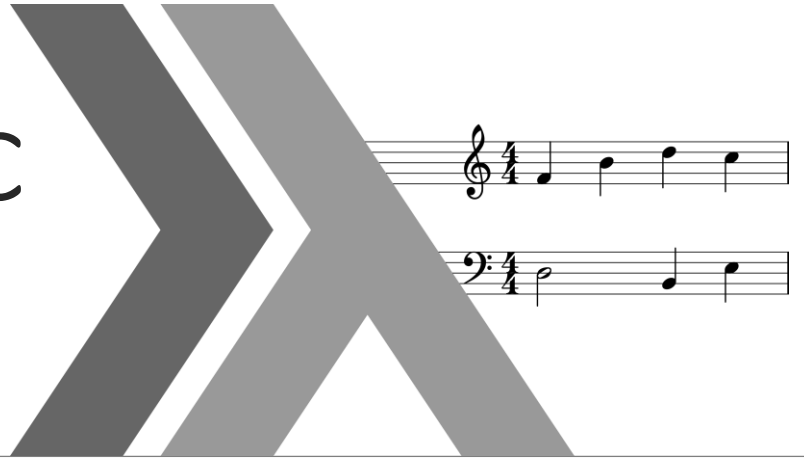
Euterpea's Music Types: Modifiers



DONYA QUICK

http://www.euterpea.com

Prerequisites

- Have Haskell and Euterpea installed.
- Be familiar with Euterpea's Music types.
  - Pitches, notes, rests, sequential and parallel composition, etc.
- Tutorials on these prerequisites are available here: http://www.euterpea.com/tutorials

In This Tutorial

- The `Modify` constructor for Euterpea's Music type.
  - Tempo changes
  - Transposition
  - Setting instruments
- Using modifiers to create two create a round and a phase composition.

Euterpea's `Music` Type

```
data Music a = Prim (Primitive a)
    | (Music a) :+: (Music a)
    | (Music a) :=: (Music a)
    | Modify Control (Music a)
```

sequential composition

parallel composition

for tempo changes, etc.

- The `Modify` constructor is used to provide interpretive contexts for musical subtrees.
- The `Control` type is used to stipulate what type of modification should take place.
- **Very important:** Modify nodes do NOT affect their subtrees' data. They simply alter the performance.

The Control Data Type

```
data Control =
     Tempo       Rational
   | Transpose   AbsPitch
   | Instrument  InstrumentName
   | Phrase      [PhraseAttribute]
   | KeySig      PitchClass Mode
   | Custom      String
  deriving (Show, Eq, Ord)
```

The Control data type has several constructors for providing interpretive context to musical subtrees.

The Control Data Type: `Tempo`

```
data Control =
      Tempo          Rational
    | Transpose      AbsPitch
    | Instrument     InstrumentName
    | Phrase         [PhraseAttribute]
    | KeySig         PitchClass Mode
    | Custom         String
  deriving (Show, Eq, Ord)
```

Tempo changes stipulate a relative playback rate change. A value of 1 means no change, while a value of 2 means play twice as fast and 1/2 is half the original speed.

The durations of the modified Music subtree are *not* altered. You will only hear the tempo change at playback.

When writing MIDI files with `writeMidi`, the durations of notes *are* altered – it does *not* insert a tempo change event. This means a half note becomes a quarter note with a tempo change of 2.

The Control Data Type: `Transpose`

```
data Control =
       Tempo       Rational
     | Transpose   AbsPitch
     | Instrument  InstrumentName
     | Phrase      [PhraseAttribute]
     | KeySig      PitchClass Mode
     | Custom      String
   deriving (Show, Eq, Ord)
```

Transposition interprets the subtree offset up or down from its original pitches. The amount is specified in semitones or half steps. A value of 5 means to transpose up 5 half steps, while -2 means down a whole step (2 half steps).

Again, `Transpose` does not alter the pitches in the subtree. It only affects the pitch you hear upon playback.

When writing MIDI files, notes will be written with the altered pitch. So, if a note with pitch number 60 has a `Transpose 5` modifier above it in the tree, the resulting MIDI pitch number will be 65.

The Control Data Type: `Instrument`

```
data Control =
     Tempo       Rational
   | Transpose   AbsPitch
   | Instrument  InstrumentName
   | Phrase      [PhraseAttribute]
   | KeySig      PitchClass Mode
   | Custom      String
  deriving (Show, Eq, Ord)
```

Instrument changes set the instrument for a musical subtree.

To see a list of `InstrumentName` values you can use, type **`:i InstrumentName`** into GHCi. Some examples of instrument names include:

AcousticGrandPiano
Marimba
FretlessBass
PizzicatoStrings
etc.

Common Problem: Multiple Instrument Changes

- The innermost instrument assignment is what you will hear.
- If you do this:

```
m1 = Instrument Marimba (c 4 qn)
m2 = Instrument Flute m1
```

  you will still hear a marimba for m2, not a flute!

- To change the instrument of a section that already has an instrument assigned, use the `removeInstruments` function on it first:

```
m2 = Instrument Flute $
     removeInstruments m1
```

Haskell syntax: `f $ g x` is equivalent to `f (g x)`.

The Control Data Type: `Phrase`

```
data Control =
     Tempo       Rational
   | Transpose   AbsPitch
   | Instrument  InstrumentName
   | Phrase      [PhraseAttribute]
   | KeySig      PitchClass Mode
   | Custom      String
  deriving (Show, Eq, Ord)
```

Phrase annotations include crescendos, decrescendos, accents, staccato, legato, and so on.

Phrase modifiers are the subject of their own tutorial.

The Control Data Type: `KeySig`

```
data Control =
     Tempo       Rational
   | Transpose   AbsPitch
   | Instrument  InstrumentName
   | Phrase      [PhraseAttribute]
   | KeySig      PitchClass Mode
   | Custom      String
  deriving (Show, Eq, Ord)
```

Key signatures can be annotated through the use of a root and mode, but they do not affect playback or MIDI file export. They exist to support alternate file exporting algorithms (not included with Euterpea) and user-made algorithms that may benefit from key annotations.

The Control Data Type: `KeySig`

```
data Control =
     Tempo        Rational
   | Transpose    AbsPitch
   | Instrument   InstrumentName
   | Phrase       [PhraseAttribute]
   | KeySig       PitchClass Mode
   | Custom       String
  deriving (Show, Eq, Ord)
```

Custom modifiers do not affect anything about playback or MIDI file export. They exist as a way for users to add their own modifiers for use in their own algorithms.

Modifier Shorthands

- ```
  tempo :: Dur -> Music a -> Music a
  tempo r m = Modify (Tempo r) m
  ```
- ```
  transpose :: AbsPitch -> Music a -> Music a
  transpose i m = Modify (Transpose i) m
  ```
- ```
  instrument :: InstrumentName -> Music a -> Music a
  instrument i m = Modify (Instrument i) m
  ```
- ```
  phrase :: [PhraseAttribute] -> Music a -> Music a
  phrase pa m = Modify (Phrase pa) m
  ```

Using these functions can result in shorter code than writing out the `Modify` statements manually.

Example: Creating a Round

```
twinkle :: Music Pitch
twinkle = line [c 5 qn, c 5 qn, g 5 qn,
  g 5 qn, a 5 qn, a 5 qn, g 5 hn,
  f 4 qn, f 4 qn, e 4 qn, e 4 qn,
  d 4 qn, d 4 qn, c 4 hn]

twinkle4 =
  line [twinkle, twinkle, twinkle, twinkle]

twinkleRound =
  instrument Marimba twinkle4 :=:
  instrument Vibraphone (rest wn :+: twinkle4) :=:
  instrument Celesta (rest 2 :+: twinkle4)
```

A **round** in a type of composition where each instrument plays the same thing, but each one comes in delayed by a different amount.

Here we define the first four measures of "Twinkle Twinkle Little Star," repeat the melody four times, and then create a round for three different instruments.

Example: a Simple Phase Composition

```
m1 = c 4 en :+: rest en :+: d 4 en :+: rest en :+: m1
m2 = Modify (Transpose 3) (Modify (Tempo 1.01) m1)
m3 = Modify (Transpose 5) (Modify (Tempo 1.02) m1)
phase = m1 :=: m2 :=: m3
```

A **phase composition** involves two or more parts, each playing the same repeating, simple motif. The parts play at very slightly different tempos, creating gradually changing rhythmic patterns over time. Phase compositions are very difficult for human performers, but easy for a computer!

Run `play phase` in GHCi and listen to the rhythmic patterns evolve over time. Use Ctrl+C to stop the playback.

Notice that `m1` is an infinite structure due to its recursive definition. This means that `m2`, `m3`, and `phase` are also infinite. To write a MIDI file of an infinite structure, you must use `cut` to preserve only a finite amount. For example, we can write the first 100 measures to a MIDI file by doing this:

```
writeMidi "phase.mid" (cut 100 phase)
```

More Examples and Information

- More examples:
  euterpea.com/examples/

- Euterpea API and quick references:
  euterpea.com/api/

- Other Tutorials
  euterpea.com/tutorials