HarmLang

A language for music notation, manipulation, and probabilistic analysis

Existing Languages/Tools

- Finale/Sibelius
- LilyPond
- Haskore
- Euterpea

Goals

- **Simplicity** and **conciseness** of notation
- Powerful **type system** and **initial basis**
- **Programmatic transformation** of music
- Strong **Haskell integration**
- Integration with **MIDI**
- **Probabilistic** analysis (and generation) of music

Target Audience

Musicians

Notate Compositions

Generate Arrangements

Transpose & Transform

Programmers

Write groovy probabilistic programs.

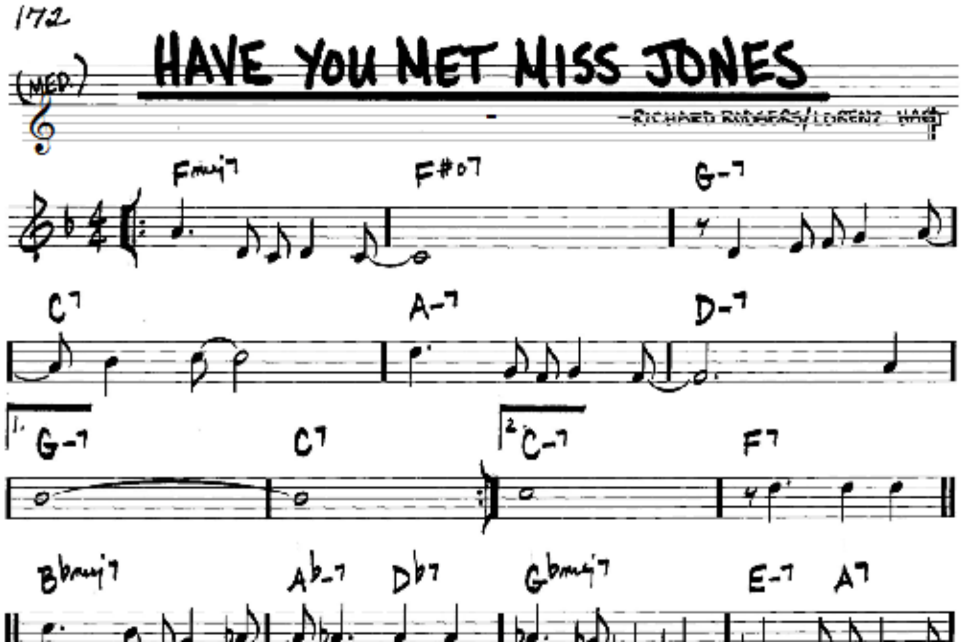
Leverage Haskell libraries

Bask in the glory of the HarmLang type system

POWER USERS!

Notational Syntax: Motivation

Excerpt from the Sixth Edition Real Book



Scores

- + Look great!
- + Universally understood by Western musicians
- + Many years of precedent
- - Not a text based format
- - Limited modularity
- - Low information density
- - Ambiguity abounds
  - - Sometimes human interpretation is required: not good for a computer program!

BIAS WARNING

The following slide may be slightly biased. The authors have used Lilypond, and found it to be a “mixed bag.”

Notational Syntax: Motivation

Lilypond

```
\chordmode {
  \startChords
  \startSong
  \myMark "A"
  \startPart
  f1:maj7 | fis:dim7 | g:m7 | c:7 | \myEndLine
  a:m7 | d:m7 | g:m7 | c:7 | \myEndLine
  \endPart
  \myMark "A"
  \startPart
  f:maj7 | fis:dim7 | g:m7 | c:7 | \myEndLine
  a:m7 | d:m7 | c:m7 | f:7 | \myEndLine
  \endPart
  \myMark "B"
  \startPart
  bes:maj7 | aes2:m7 des:7 | ges1:maj7 | e2:m7 a:7 | \myEndLine
  d1:maj7 | aes2:m7 des:7 | ges1:maj7 | g2:m7 c:7 | \myEndLine
  \endPart
  \myMark "A"
  \startPart
  \repeat volta 2 {
      f1:maj7 | fis:dim7 | g:m7 | c2:7 bes:7 | \myEndLine
      a:m7 d:7.9- | g:m7 c:7 |
  } \alternative {
      {
      f d:m7 | g:m7 c:7 |
```

- + Some parts vaguely resemble traditional music notation.
- + High learning curve prevents too many casual users from clogging up the message boards.
- - Terrible "pseudodeclarative" syntax
- - Confusing music notation syntax
  - - "des" represents the note Db
- - Lots of commands and literals.
- - Verbose (doesn't fit on screen)

Sauce: goo.gl/dfisrc

HarmLang Syntax

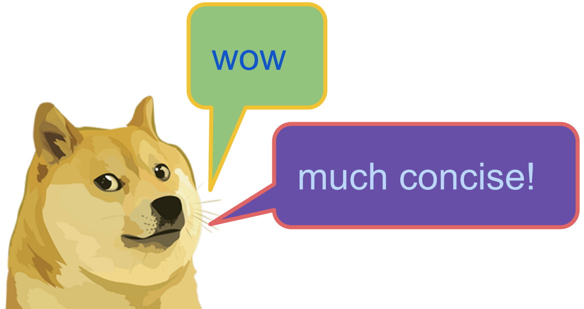
```
a1 = [hl|[FMa7:4 F#o7:4 Gm7:4 C7:4 Am7:4 Dm7:4
Gm7:4 C7:4]|]

a2 = [hl|[FMa7:4 F#o7:4 Gm7:4 C7:4 Am7:4 Dm7:4
Cm7:4 F7:4]|]

b  = [hl|[BbMa7:4 Abm7:2 Db7:2 GbMa7:4 Em7:2 A7:2
DMa7:4 Abm7:2 Db7:2 GbMa7:4 Gm7:2 C7:2]|]

a3 = [hl|[FMa7:4 F#o7:4 Gm7:4 C7:2 Bb7:2 Am7:4
D7:4 Gm7:4 C7:4 FMa7:4 Gm7:4 C7:4]|]

progression = a1 ++ a2 ++ b ++ a3
```



- + “Completely”™ hides confusing Haskell types.
- + Extremely concise.
- + Logically named entities are familiar to musicians.
- + Syntax is very familiar to Haskell programmers, but not alien to functional or imperative programmers.
- + Lilypond is to HarmLang as LaTex is to (a non sucky) Slideshow
- - There are no cons. HarmLang is perfect. Do not believe otherwise.

Features and Abstractions

- Simple and concise music notation format
- Rich but intuitive type system
  - Haskell features are seamlessly abstracted:
    - [Chord] is ChordProgression
    - `show ChordProgression` is “overridden” to mimic HarmLang syntax
  - ChordDistribution is a fully
- Integration with MIDI
- Probabilistic analysis support
- Interpreted AND compiled HarmLang syntax
  - Compile time syntax through Quasiquotation.
  - Runtime syntax through “interpret” functions.

Implementation

- Built in Haskell as an embedded DSL
- HarmLang literals are quasiquoted with intuitive syntax.
    - Interval
    - PitchClass
    - Pitch
    - Note
    - Chord
    - TimedChord
    - PitchProgression
    - ChordProgression
    - TimedChordProgression
    - NoteProgression
- Enormous initial basis!
- HarmLang types allow useful operations.
    - Typeclasses and associated functions.

Libraries and Language Extensions

- Language Extensions:
  - Some ‘embedded languages” have very poor syntax and don’t mesh well with the host language.
    - HarmLang doesn’t have this problem!

```
{-# LANGUAGE QuasiQuotes #-}
{-# LANGUAGE DeriveDataTypeable #-}
{-# LANGUAGE FlexibleInstances #-}
{-# LANGUAGE OverlappingInstances #-}
{-# LANGUAGE TypeSynonymInstances #-}
```

- Libraries:
  - HUnit, Parsec, syb, template-haskell (Obviously)
  - Codec.Midi
  - containers

HarmLang Literals

```
[hl| [A@5 C#@6] |]
```

⇨

```
[Pitch (PitchClass 0) (Octave 5), Pitch
(PitchClass 4) (Octave 6)]
```

```
[[hl|'AM:4'|], [hl|'C#m7b5:7'|]]
```

⇨

```
[TimedChord (Harmony (PitchClass 0) [(Interval 4),
(Interval 7)]) (Time 4 4), TimedChord (Harmony
(PitchClass 4) [(Interval 3), (Interval 6),
(Interval 10)]) (Time 7 4)]
```

```
[hl| [B@5:3 D@6:3/2] |]
```

⇨

```
[Note (Pitch (PitchClass 2) (Octave 5)) (Time 3
4), Note (Pitch (PitchClass 5) (Octave 6)) (Time 3
2)]
```

HarmLang Typeclasses in Action

```
--Enum typeclass (implemented for all finite enumerable types)
allChords :: [Chord]
allChords = [[hl| 'A[]' |]..[hl| 'G#[1 2 3 4 5 6 7 8 9 10 11]' |]]

--Transposable typeclass
transposeBlues :: Interval -> ChordProgression
transposeBlues interval = transpose interval [hl| [CM C7 F7 C7 G7 F7 C7 G7] |]

--Eq?  That works too
aTrueValue :: Bool
aTrueValue = (==) (transpose [hl| '2' |] [hl| [Cm7 F7 Bbma7] |]) [hl| [Dm7 G7 Cma7] |]

--MIDIable.  Because music wants to be listened to.
outputToMidi [hl| [A@4:1/4 B@4:1/4 C@4:1/4 D@4:1/4 E@4:1/4 F@4:1/4 G@4:1/4] |] "afile.mid"

--Show is implemented too.  In fact, the output of any show is valid HarmLang!
interpretChordProgression $ show [hl | [G7 F7 C7] |]
```

Other useful functions

Sorry about these function names. They are terrible! We're working on it.

```
inverse :: Interval -> Interval
--The function such that (transpose a (transpose (inverse a) b) = b.

toChord :: PitchClass -> [PitchClass] -> Chord
getNotesFromChord :: Chord -> [PitchClass]
--There are many ways to represent a chord, some containing more information
than others.
--We need a way to convert between representations.

toTimedProgression :: Time -> ChordProgression -> TimedChordProgression
toUntimedProgression :: TimedChordProgression -> ChordProgression
--Sometimes we need to add or remove timing information from a progression.

chordInversions :: Chord -> [Chord]
--Yields the musical inversions of a chord.
```

Demo

Future Work

- Monads are confusing! But HarmLang users have to use the IO monad if they want output (which they probably do).
  - Can be thought of as a bit of syntactic weirdness, and used without a full understanding of the forces at play.
- We wanted probabilistic generation of music. But…
  - Generation $\rightarrow$ sampling, sampling $\rightarrow$ entropy, entropy $\rightarrow$ impurity, impurity $\rightarrow$ monads, monads $\rightarrow$ headaches $\Rightarrow$ generation $\rightarrow$ headaches.
- Antiquotation for pattern matching.

Questions?