Advanced Functional Programming in Industry

José Pedro Magalhães

chordify®

January 23, 2015
Berlin, Germany

Introduction

- Haskell: a statically typed, lazy, purely functional language
- Modelling musical harmony using Haskell
- Applications of a model of harmony:
  - Musical analysis
  - Finding cover songs
  - Generating chords and melodies
  - Correcting errors in chord extraction from audio sources
  - Chordify—a web-based music player with chord recognition

Demo: Chordify

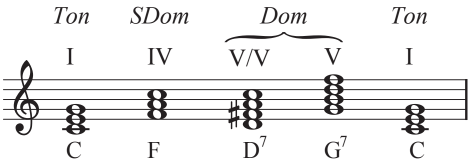
Demo:

chordify®

`http://chordify.net`

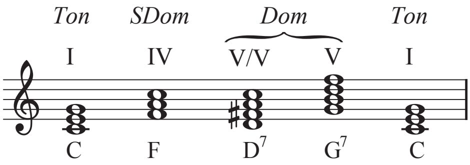
Table of Contents

What is harmony?



- Harmony arises when at least two notes sound at the same time
- Harmony induces tension and release patterns, that can be described by music theory and music cognition
- The internal structure of the chord has a large influence on the consonance or dissonance of a chord
- The surrounding context also has a large influence

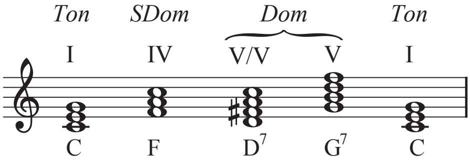
What is harmony?

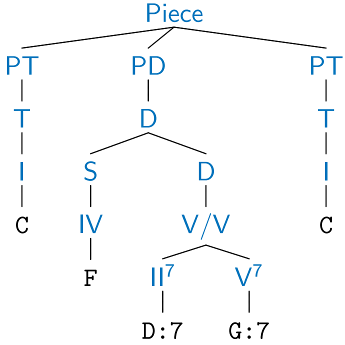


- Harmony arises when at least two notes sound at the same time
- Harmony induces tension and release patterns, that can be described by music theory and music cognition
- The internal structure of the chord has a large influence on the consonance or dissonance of a chord
- The surrounding context also has a large influence

Demo: how harmony affects melody

An example harmonic analysis





Why are harmony models useful?

Having a model for musical harmony allows us to automatically determine the functional meaning of chords in the tonal context.
The model determines which chords “fit” on a particular moment in a song.

Why are harmony models useful?

Having a model for musical harmony allows us to automatically determine the functional meaning of chords in the tonal context. The model determines which chords “fit” on a particular moment in a song. This is useful for:

- Musical information retrieval (find songs similar to a given song)
- Audio and score recognition (improving recognition by knowing which chords are more likely to appear)
- Music generation (create sequences of chords that conform to the model)

Table of Contents

Why Haskell?

Haskell is a strongly-typed pure functional programming language:

Strongly-typed
: All values are classified by their type, and types are known at compile time (statically). This gives us strong guarantees about our code, avoiding many common mistakes.

Pure
: There are no side-effects, so Haskell functions are like mathematical functions.

Functional
: A Haskell program is an expression, not a sequence of statements. Functions are first class citizens, and explicit state is avoided.

Notes

```
data Root   = A | B | C | D | E | F | G
type Octave = Int
data Note   = Note Root Octave
```

Notes

```
data Root   = A | B | C | D | E | F | G
type Octave = Int
data Note   = Note Root Octave

a4, b4, c4, d4, e4, f4, g4 :: Note
a4 = Note A 4
b4 = Note B 4
c4 = Note C 4
d4 = Note D 4
e4 = Note E 4
f4 = Note F 4
g4 = Note G 4
```

Melody

type Melody = [Note]

cMajScale :: Melody
cMajScale = [c4, d4, e4, f4, g4, a4, b4]

Melody

```
type Melody = [Note]

cMajScale :: Melody
cMajScale = [c4, d4, e4, f4, g4, a4, b4]

cMajScaleRev :: Melody
cMajScaleRev = reverse cMajScale
```

Melody

```
type Melody = [Note]

cMajScale :: Melody
cMajScale = [c4, d4, e4, f4, g4, a4, b4]

cMajScaleRev :: Melody
cMajScaleRev = reverse cMajScale

reverse :: [α] → [α]
reverse []      = []
reverse (h : t) = reverse t ++ [h]
(++) :: [α] → [α] → [α]
(++) = ...
```

Transposition

Transposing a melody one octave higher:

```
octaveUp :: Octave → Octave
octaveUp n = n + 1

noteOctaveUp :: Note → Note
noteOctaveUp (Note r o) = Note r (octaveUp o)

melodyOctaveUp :: Melody → Melody
melodyOctaveUp m = map noteOctaveUp m
```

Generation, analysis

Building a repeated melodic phrase:

$$\text{ostinato} :: \text{Melody} \rightarrow \text{Melody}$$
$$\text{ostinato } m = m +\!\!+ \text{ ostinato } m$$

Generation, analysis

Building a repeated melodic phrase:

$$\text{ostinato} :: \text{Melody} \rightarrow \text{Melody}$$
$$\text{ostinato m} = \text{m} + \text{ostinato m}$$

Is a given melody in C major?

$$\text{root} :: \text{Note} \rightarrow \text{Root}$$
$$\text{root (Note r o)} = \text{r}$$

$$\text{isCMaj} :: \text{Melody} \rightarrow \text{Bool}$$
$$\text{isCMaj} = \text{all} \ (\in \text{cMajScale}) \circ \text{map root}$$

“Details” left out

We have seen only a glimpse of music representation in Haskell.

- Rhythm
- Accidentals
- Intervals
- Voicing
- ...

A good pedagogical reference on using Haskell to represent music:
`http://di.uminho.pt/~jno/html/ipm-1011.html`

A serious library for music manipulation:
`http://www.haskell.org/haskellwiki/Haskore`

Table of Contents

Application: harmony analysis

Parsing the sequence G_{min} C^7 G_{min} C^7 F_{Maj} D^7 G^7 C_{Maj}:

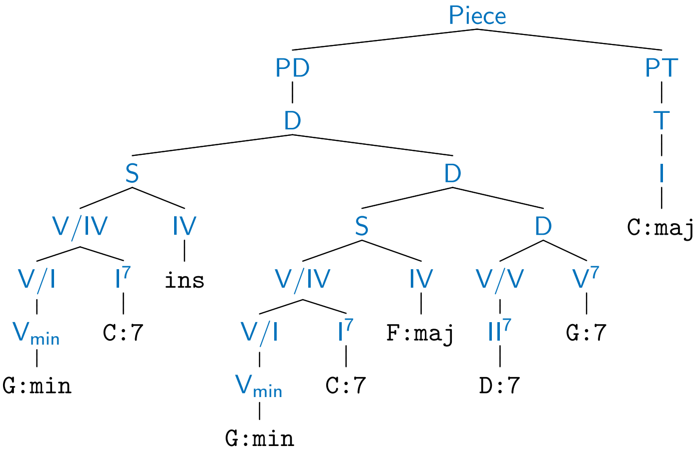


Table of Contents

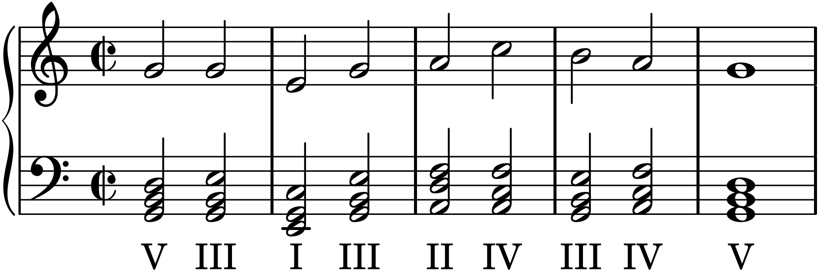
Application: harmonic similarity

- A practical application of a harmony model is to estimate harmonic similarity between songs
- The more similar the trees, the more similar the harmony
- We don't want to write a diff algorithm for our complicated model; we get it automatically by using a *generic diff*
- The generic diff is a type-safe tree-diff algorithm, part of a student's MSc work at Utrecht University
- Generic, thus working for any model, and independent of changes to the model

Table of Contents

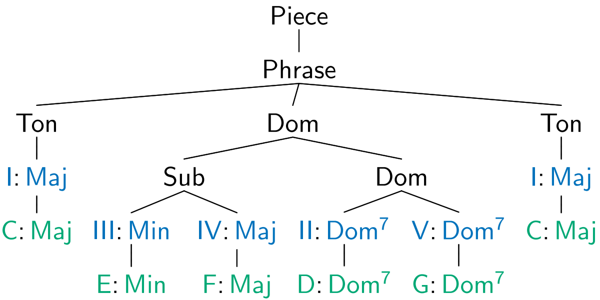
Application: automatic harmonisation of melodies

Another practical application of a harmony model is to help selecting good harmonisations (chord sequences) for a given melody:



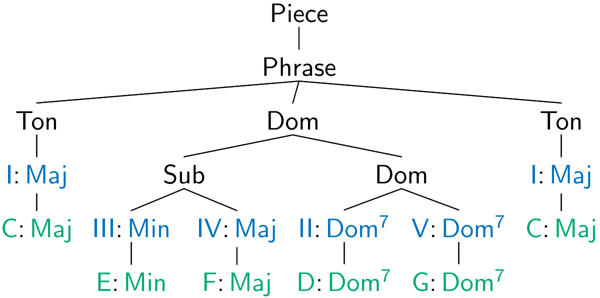
We generate candidate chord sequences, parse them with the harmony model, and select the one with the least errors.

Visualising harmonic structure



You can see this tree as having been produced by taking the chords in green as input...

Generating harmonic structure



You can see this tree as having been produced by taking the chords in green as input. . . or the chords might have been dictated by the structure!

A functional model of harmony

$$\text{Piece}_{\mathfrak{M}} \rightarrow [\text{Phrase}_{\mathfrak{M}}] \qquad (\mathfrak{M} \in \{\text{Maj}, \text{Min}\})$$

A functional model of harmony

$$\text{Piece}_{\mathfrak{M}} \rightarrow [\text{Phrase}_{\mathfrak{M}}] \qquad (\mathfrak{M} \in \{\text{Maj}, \text{Min}\})$$

$$\begin{aligned} \text{Phrase}_{\mathfrak{M}} \rightarrow\ & \text{Ton}_{\mathfrak{M}}\ \text{Dom}_{\mathfrak{M}}\ \text{Ton}_{\mathfrak{M}} \\ \mid\ & \text{Dom}_{\mathfrak{M}}\ \text{Ton}_{\mathfrak{M}} \end{aligned}$$

A functional model of harmony

$$\text{Piece}_{\mathfrak{M}} \rightarrow [\text{Phrase}_{\mathfrak{M}}] \qquad (\mathfrak{M} \in \{\text{Maj}, \text{Min}\})$$

$$\begin{aligned}\text{Phrase}_{\mathfrak{M}} &\rightarrow \text{Ton}_{\mathfrak{M}}\ \text{Dom}_{\mathfrak{M}}\ \text{Ton}_{\mathfrak{M}} \\ &\mid \qquad\quad \text{Dom}_{\mathfrak{M}}\ \text{Ton}_{\mathfrak{M}}\end{aligned}$$

$$\text{Ton}_{\text{Maj}} \rightarrow \text{I}_{\text{Maj}}$$
$$\text{Ton}_{\text{Min}} \rightarrow \text{I}^{m}_{\text{Min}}$$

A functional model of harmony

$$\text{Piece}_{\mathfrak{M}} \rightarrow [\text{Phrase}_{\mathfrak{M}}] \qquad (\mathfrak{M} \in \{\text{Maj}, \text{Min}\})$$

$$\begin{aligned}\text{Phrase}_{\mathfrak{M}} &\rightarrow \text{Ton}_{\mathfrak{M}}\ \text{Dom}_{\mathfrak{M}}\ \text{Ton}_{\mathfrak{M}} \\ &\mid \quad \text{Dom}_{\mathfrak{M}}\ \text{Ton}_{\mathfrak{M}}\end{aligned}$$

$$\text{Ton}_{\text{Maj}} \rightarrow \text{I}_{\text{Maj}}$$

$$\text{Ton}_{\text{Min}} \rightarrow \text{I}^{m}_{\text{Min}}$$

$$\begin{aligned}\text{Dom}_{\mathfrak{M}} &\rightarrow \text{V}^{7}_{\mathfrak{M}} \\ &\mid \text{Sub}_{\mathfrak{M}}\ \text{Dom}_{\mathfrak{M}} \\ &\mid \text{II}^{7}_{\mathfrak{M}}\ \text{V}^{7}_{\mathfrak{M}}\end{aligned}$$

A functional model of harmony

$$\text{Piece}_{\mathfrak{M}} \rightarrow [\text{Phrase}_{\mathfrak{M}}] \qquad (\mathfrak{M} \in \{\text{Maj}, \text{Min}\})$$

$$\begin{aligned}\text{Phrase}_{\mathfrak{M}} &\rightarrow \text{Ton}_{\mathfrak{M}}\ \text{Dom}_{\mathfrak{M}}\ \text{Ton}_{\mathfrak{M}} \\ &\mid \quad \text{Dom}_{\mathfrak{M}}\ \text{Ton}_{\mathfrak{M}}\end{aligned}$$

$$\text{Ton}_{\text{Maj}} \rightarrow \text{I}_{\text{Maj}}$$

$$\text{Ton}_{\text{Min}} \rightarrow \text{I}^{m}_{\text{Min}}$$

$$\begin{aligned}\text{Dom}_{\mathfrak{M}} &\rightarrow \text{V}^{7}_{\mathfrak{M}} \\ &\mid \text{Sub}_{\mathfrak{M}}\ \text{Dom}_{\mathfrak{M}} \\ &\mid \text{II}^{7}_{\mathfrak{M}}\ \text{V}^{7}_{\mathfrak{M}}\end{aligned}$$

$$\begin{aligned}\text{Sub}_{\text{Maj}} &\rightarrow \text{II}^{m}_{\text{Maj}} \\ &\mid \text{IV}_{\text{Maj}} \\ &\mid \text{III}^{m}_{\text{Maj}}\ \text{IV}_{\text{Maj}}\end{aligned}$$

$$\text{Sub}_{\text{Min}} \rightarrow \text{IV}^{m}_{\text{Min}}$$

A functional model of harmony

$$\text{Piece}_{\mathfrak{M}} \rightarrow [\text{Phrase}_{\mathfrak{M}}] \qquad (\mathfrak{M} \in \{\text{Maj}, \text{Min}\})$$

$$\begin{aligned} \text{Phrase}_{\mathfrak{M}} \rightarrow\ & \text{Ton}_{\mathfrak{M}}\ \text{Dom}_{\mathfrak{M}}\ \text{Ton}_{\mathfrak{M}} \\ \mid\ & \text{Dom}_{\mathfrak{M}}\ \text{Ton}_{\mathfrak{M}} \end{aligned}$$

$$\text{Ton}_{\text{Maj}} \rightarrow \text{I}_{\text{Maj}}$$
$$\text{Ton}_{\text{Min}} \rightarrow \text{I}^{m}_{\text{Min}}$$

$$\begin{aligned} \text{Dom}_{\mathfrak{M}} \rightarrow\ & \text{V}^{7}_{\mathfrak{M}} \\ \mid\ & \text{Sub}_{\mathfrak{M}}\ \text{Dom}_{\mathfrak{M}} \\ \mid\ & \text{II}^{7}_{\mathfrak{M}}\ \text{V}^{7}_{\mathfrak{M}} \end{aligned}$$

$$\begin{aligned} \text{Sub}_{\text{Maj}} \rightarrow\ & \text{II}^{m}_{\text{Maj}} \\ \mid\ & \text{IV}_{\text{Maj}} \\ \mid\ & \text{III}^{m}_{\text{Maj}}\ \text{IV}_{\text{Maj}} \end{aligned}$$

$$\text{Sub}_{\text{Min}} \rightarrow \text{IV}^{m}_{\text{Min}}$$

Simple, but enough for now, *and easy to extend*.

Now in Haskell—I

A naive datatype encoding musical harmony:

```
data Piece = Piece [Phrase]

data Phrase where
  Phrase_IVI :: Ton → Dom → Ton → Phrase
  Phrase_VI  ::       Dom → Ton → Phrase
```

Now in Haskell—I

A naive datatype encoding musical harmony:

$$\textbf{data}\ \text{Piece} = \text{Piece}\ [\,\text{Phrase}\,]$$

$$\begin{aligned}
&\textbf{data}\ \text{Phrase}\ \textbf{where}\\
&\quad \text{Phrase}_{\text{IVI}} :: \text{Ton} \to \text{Dom} \to \text{Ton} \to \text{Phrase}\\
&\quad \text{Phrase}_{\text{VI}}\ \ :: \text{Dom} \to \text{Ton} \to \text{Phrase}
\end{aligned}$$

$$\begin{aligned}
&\textbf{data}\ \text{Ton}\ \textbf{where}\\
&\quad \text{Ton}_{\text{Maj}} :: \text{Degree} \to \text{Ton}\\
&\quad \text{Ton}_{\text{Min}} :: \text{Degree} \to \text{Ton}
\end{aligned}$$

Now in Haskell—I

A naive datatype encoding musical harmony:

$$\mathbf{data}\ \mathsf{Piece} = \mathsf{Piece}\ [\mathsf{Phrase}]$$

$$\begin{aligned}
&\mathbf{data}\ \mathsf{Phrase}\ \mathbf{where}\\
&\quad \mathsf{Phrase}_{\mathsf{IVI}} :: \mathsf{Ton} \to \mathsf{Dom} \to \mathsf{Ton} \to \mathsf{Phrase}\\
&\quad \mathsf{Phrase}_{\mathsf{VI}}\ :: \mathsf{Dom} \to \mathsf{Ton} \to \mathsf{Phrase}
\end{aligned}$$

$$\begin{aligned}
&\mathbf{data}\ \mathsf{Ton}\ \mathbf{where}\\
&\quad \mathsf{Ton}_{\mathsf{Maj}} :: \mathsf{Degree} \to \mathsf{Ton}\\
&\quad \mathsf{Ton}_{\mathsf{Min}} :: \mathsf{Degree} \to \mathsf{Ton}
\end{aligned}$$

$$\begin{aligned}
&\mathbf{data}\ \mathsf{Dom}\ \mathbf{where}\\
&\quad \mathsf{Dom}_{\mathsf{V^7}} :: \mathsf{Degree} \to \mathsf{Dom}\\
&\quad \mathsf{Dom}_{\mathsf{IV-V}} :: \mathsf{SDom} \to \mathsf{Dom} \to \mathsf{Dom}\\
&\quad \mathsf{Dom}_{\mathsf{II-V}} :: \mathsf{Degree} \to \mathsf{Degree} \to \mathsf{Dom}
\end{aligned}$$

Now in Haskell—I

A naive datatype encoding musical harmony:

$$
\begin{array}{l}
\textbf{data}\ \mathsf{Piece} = \mathsf{Piece}\ [\,\mathsf{Phrase}\,] \\[1ex]
\textbf{data}\ \mathsf{Phrase}\ \textbf{where} \\
\quad \mathsf{Phrase_{IVI}} :: \mathsf{Ton} \to \mathsf{Dom} \to \mathsf{Ton} \to \mathsf{Phrase} \\
\quad \mathsf{Phrase_{VI}}\ \ :: \mathsf{Dom} \to \mathsf{Ton} \to \mathsf{Phrase} \\[1ex]
\textbf{data}\ \mathsf{Ton}\ \textbf{where} \\
\quad \mathsf{Ton_{Maj}} :: \mathsf{Degree} \to \mathsf{Ton} \\
\quad \mathsf{Ton_{Min}} :: \mathsf{Degree} \to \mathsf{Ton} \\[1ex]
\textbf{data}\ \mathsf{Dom}\ \textbf{where} \\
\quad \mathsf{Dom_{V^7}} :: \mathsf{Degree} \to \mathsf{Dom} \\
\quad \mathsf{Dom_{IV-V}} :: \mathsf{SDom} \to \mathsf{Dom} \to \mathsf{Dom} \\
\quad \mathsf{Dom_{II-V}} :: \mathsf{Degree} \to \mathsf{Degree} \to \mathsf{Dom} \\[1ex]
\textbf{data}\ \mathsf{Degree} = \mathsf{I} \mid \mathsf{II} \mid \mathsf{III} \ldots
\end{array}
$$

Now in Haskell—II

A GADT encoding musical harmony:

$\textbf{data}\ \text{Mode} = \text{Maj}_{\text{Mode}} \mid \text{Min}_{\text{Mode}}$

$\textbf{data}\ \text{Piece}\ (\mu :: \text{Mode})\ \textbf{where}$
$\quad \text{Piece} :: [\,\text{Phrase}\ \mu\,] \rightarrow \text{Piece}\ \mu$

Now in Haskell—II

A GADT encoding musical harmony:

$$\textbf{data}\ \mathsf{Mode} = \mathsf{Maj}_{\mathsf{Mode}} \mid \mathsf{Min}_{\mathsf{Mode}}$$

$$\begin{aligned}
&\textbf{data}\ \mathsf{Piece}\ (\mu :: \mathsf{Mode})\ \textbf{where}\\
&\quad \mathsf{Piece} :: [\,\mathsf{Phrase}\ \mu\,] \to \mathsf{Piece}\ \mu
\end{aligned}$$

$$\begin{aligned}
&\textbf{data}\ \mathsf{Phrase}\ (\mu :: \mathsf{Mode})\ \textbf{where}\\
&\quad \mathsf{Phrase}_{\mathsf{IVI}} :: \mathsf{Ton}\ \mu \to \mathsf{Dom}\ \mu \to \mathsf{Ton}\ \mu \to \mathsf{Phrase}\ \mu\\
&\quad \mathsf{Phrase}_{\mathsf{VI}}\ \ :: \mathsf{Dom}\ \mu \to \mathsf{Ton}\ \mu \to \mathsf{Phrase}\ \mu
\end{aligned}$$

Now in Haskell—II

A GADT encoding musical harmony:

$$\textbf{data}\ \mathsf{Mode} = \mathsf{Maj}_{\mathsf{Mode}} \mid \mathsf{Min}_{\mathsf{Mode}}$$

$$\textbf{data}\ \mathsf{Piece}\ (\mu :: \mathsf{Mode})\ \textbf{where}$$
$$\quad \mathsf{Piece} :: [\,\mathsf{Phrase}\ \mu\,] \rightarrow \mathsf{Piece}\ \mu$$

$$\textbf{data}\ \mathsf{Phrase}\ (\mu :: \mathsf{Mode})\ \textbf{where}$$
$$\quad \mathsf{Phrase}_{\mathsf{IVI}} :: \mathsf{Ton}\ \mu \rightarrow \mathsf{Dom}\ \mu \rightarrow \mathsf{Ton}\ \mu \rightarrow \mathsf{Phrase}\ \mu$$
$$\quad \mathsf{Phrase}_{\mathsf{VI}} :: \mathsf{Dom}\ \mu \rightarrow \mathsf{Ton}\ \mu \rightarrow \mathsf{Phrase}\ \mu$$

$$\textbf{data}\ \mathsf{Ton}\ (\mu :: \mathsf{Mode})\ \textbf{where}$$
$$\quad \mathsf{Ton}_{\mathsf{Maj}} :: \mathsf{SD}\ \mathsf{I}\ \mathsf{Maj} \rightarrow \mathsf{Ton}\ \mathsf{Maj}_{\mathsf{Mode}}$$
$$\quad \mathsf{Ton}_{\mathsf{Min}} :: \mathsf{SD}\ \mathsf{I}\ \mathsf{Min} \rightarrow \mathsf{Ton}\ \mathsf{Min}_{\mathsf{Mode}}$$

Now in Haskell—II

A GADT encoding musical harmony:

$$\textbf{data}\ \text{Mode} = \text{Maj}_{\text{Mode}} \mid \text{Min}_{\text{Mode}}$$

$$\begin{array}{l}
\textbf{data}\ \text{Piece}\ (\mu :: \text{Mode})\ \textbf{where} \\
\quad \text{Piece} :: [\,\text{Phrase}\ \mu\,] \to \text{Piece}\ \mu
\end{array}$$

$$\begin{array}{l}
\textbf{data}\ \text{Phrase}\ (\mu :: \text{Mode})\ \textbf{where} \\
\quad \text{Phrase}_{\text{IVI}} :: \text{Ton}\ \mu \to \text{Dom}\ \mu \to \text{Ton}\ \mu \to \text{Phrase}\ \mu \\
\quad \text{Phrase}_{\text{VI}}\ \ :: \text{Dom}\ \mu \to \text{Ton}\ \mu \to \text{Phrase}\ \mu
\end{array}$$

$$\begin{array}{l}
\textbf{data}\ \text{Ton}\ (\mu :: \text{Mode})\ \textbf{where} \\
\quad \text{Ton}_{\text{Maj}} :: \text{SD I Maj} \to \text{Ton}\ \text{Maj}_{\text{Mode}} \\
\quad \text{Ton}_{\text{Min}} :: \text{SD I Min} \to \text{Ton}\ \text{Min}_{\text{Mode}}
\end{array}$$

$$\begin{array}{l}
\textbf{data}\ \text{Dom}\ (\mu :: \text{Mode})\ \textbf{where} \\
\quad \text{Dom}_{\text{V}^7}\ \ \ :: \text{SD V Dom}^7 \to \text{Dom}\ \mu \\
\quad \text{Dom}_{\text{IV}-\text{V}} :: \text{SDom}\ \mu \to \text{Dom}\ \mu \to \text{Dom}\ \mu \\
\quad \text{Dom}_{\text{II}-\text{V}}\ :: \text{SD II Dom}^7 \to \text{SD V Dom}^7 \to \text{Dom}\ \mu
\end{array}$$

Now in Haskell—III

Scale degrees are the leaves of our hierarchical structure:

$$\textbf{data}\ \text{DiatonicDegree} = \text{I} \mid \text{II} \mid \text{III} \mid \text{IV} \mid \text{V} \mid \text{VI} \mid \text{VII}$$
$$\textbf{data}\ \text{Quality} = \text{Maj} \mid \text{Min} \mid \text{Dom}^{7} \mid \text{Dim}$$

$$\textbf{data}\ \text{SD}\ (\delta :: \text{DiatonicDegree})\ (\gamma :: \text{Quality})\ \textbf{where}$$
$$\quad \text{SurfaceChord} :: \text{ChordDegree} \rightarrow \text{SD}\ \delta\ \gamma$$

Now in Haskell—III

Scale degrees are the leaves of our hierarchical structure:

$$
\begin{array}{l}
\textbf{data}\ \mathsf{DiatonicDegree} = \mathsf{I} \mid \mathsf{II} \mid \mathsf{III} \mid \mathsf{IV} \mid \mathsf{V} \mid \mathsf{VI} \mid \mathsf{VII} \\
\textbf{data}\ \mathsf{Quality} = \mathsf{Maj} \mid \mathsf{Min} \mid \mathsf{Dom}^7 \mid \mathsf{Dim} \\
\textbf{data}\ \mathsf{SD}\ (\delta :: \mathsf{DiatonicDegree})\ (\gamma :: \mathsf{Quality})\ \textbf{where} \\
\quad \mathsf{SurfaceChord} :: \mathsf{ChordDegree} \to \mathsf{SD}\ \delta\ \gamma
\end{array}
$$

Now everything is properly indexed, and our GADT is effectively constrained to allow only "harmonically valid" sequences!

Generating harmony

Now that we have a datatype representing harmony sequences, how do we generate a sequence of chords?

Generating harmony

Now that we have a datatype representing harmony sequences, how do we generate a sequence of chords?

QuickCheck! We simply reuse a standard tool for generation of random test cases.

Generating harmony

Now that we have a datatype representing harmony sequences, how do we generate a sequence of chords?

QuickCheck! We simply reuse a standard tool for generation of random test cases.

And, to avoid boilerplate code once more, we use *generic programming* for generating data:

Generating harmony

Now that we have a datatype representing harmony sequences, how do we generate a sequence of chords?

QuickCheck! We simply reuse a standard tool for generation of random test cases.

And, to avoid boilerplate code once more, we use *generic programming* for generating data:

$$\mathsf{gen} :: \forall\alpha.(\mathsf{Representable}\ \alpha, \mathsf{Generate}\ (\mathsf{Rep}\ \alpha)) \Rightarrow \mathsf{Gen}\ \alpha$$

Generating harmony

Now that we have a datatype representing harmony sequences, how do we generate a sequence of chords?

QuickCheck! We simply reuse a standard tool for generation of random test cases.

And, to avoid boilerplate code once more, we use *generic programming* for generating data:

$$\text{gen} :: \forall\alpha.(\text{Representable}\ \alpha, \text{Generate}\ (\text{Rep}\ \alpha)) \Rightarrow [\,(\text{String},\text{Int})\,] \rightarrow \text{Gen}\ \alpha$$

Examples of harmony generation

```
testGen  :: Gen (Phrase MajMode)
testGen  = gen [("Dom_IV-V", 3), ("Dom_II-V", 4)]
example :: IO ()
example = let k = Key (Note ♮ C) MajMode
          in sample′ testGen >>= mapM_ (printOnKey k)
```

Examples of harmony generation

$$\begin{aligned}
&\mathsf{testGen} :: \mathsf{Gen}\ (\mathsf{Phrase}\ \mathsf{Maj}_{\mathsf{Mode}}) \\
&\mathsf{testGen} = \mathsf{gen}\ [(\texttt{"Dom_IV-V"}, 3), (\texttt{"Dom_II-V"}, 4)] \\
&\mathsf{example} :: \mathsf{IO}\ () \\
&\mathsf{example} = \mathbf{let}\ \mathsf{k} = \mathsf{Key}\ (\mathsf{Note}\ \natural\ \mathsf{C})\ \mathsf{Maj}_{\mathsf{Mode}} \\
&\qquad\quad \mathbf{in}\ \mathsf{sample}'\ \mathsf{testGen} \ggg \mathsf{mapM_}\ (\mathsf{printOnKey}\ \mathsf{k})
\end{aligned}$$

$>$ example

[C: Maj, D: Dom^7, G: Dom^7, C: Maj]
[C: Maj, G: Dom^7, C: Maj]
[C: Maj, E: Min, F: Maj, G: Maj, C: Maj]
[C: Maj, E: Min, F: Maj, D: Dom^7, G: Dom^7, C: Maj]
[C: Maj, D: Min, E: Min, F: Maj, D: Dom^7, G: Dom^7, C: Maj]

Table of Contents

Back to Chordify: chord recognition

Yet another practical application of a harmony model is to improve chord recognition from audio sources.

Chord candidates		0.92 C	0.96 E
		0.94 Gm	0.97 C
	1.00 C	1.00 G	1.00 Em
Beat number	1	2	3

How to pick the right chord from the chord candidate list? Ask the harmony model which one fits best.

Chordify: architecture

- Frontend
  - Reads user input, such as YouTube/Soundcloud/Deezer links, or files
  - Extracts audio
  - Calls the backend to obtain the chords for the audio
  - Displays the result to the user
  - Implements a queueing system, and library functionality
  - Uses PHP, JavaScript, MongoDB

Chordify: architecture

- Frontend
  - Reads user input, such as YouTube/Soundcloud/Deezer links, or files
  - Extracts audio
  - Calls the backend to obtain the chords for the audio
  - Displays the result to the user
  - Implements a queueing system, and library functionality
  - Uses PHP, JavaScript, MongoDB
- Backend
  - Takes an audio file as input, analyses it, extracts the chords
  - The chord extraction code uses GADTs, type families, generic programming (see the HarmTrace package on Hackage)
  - Performs PDF and MIDI export (using LilyPond)
  - Uses Haskell, SoX, sonic annotator, and is mostly open source

Chordify: numbers

- Online since January 2013
- Top countries: US, UK, Germany, Indonesia, Canada
- Views: 3M+ (monthly)
- Chordified songs: 1.8M+
- Registered users: 200K+

How do we handle these visitors?

- Single VPS, 6 Intel Xeon cores, 24GB RAM, 500GB SSD, 2TB hard drive
- Single server hosts both the web and database servers
- Can easily handle peaks of (at least) 700 visitors at a time
- Chordifying new songs takes some computing power, but most songs are in the database already
- Queueing system for busy periods
- Infrastructure costs are minimal

Frontend (PHP/JS) and backend (Haskell) interaction

- Frontend receives a music file, calls backend with it
- Backend computes the chords, writes them to a file:
  - ```
    1;D:min;0.232199546;0.615328798
    2;D:min;0.615328798;0.998458049
    ...
    ```
- Frontend reads this file, updates the database if necessary, and renders the result
- Backend is open-source (and GPL3); only option is to run it as a standalone executable
    ```

Logistics of an internet start-up

- Chordify is created and funded by 5 people
- If you can do without venture capital, do it!
- You might end up doing more than just functional programming, though:
  - Deciding on what features to implement next
  - Recruiting, interviewing, dealing with legal issues related to employment
  - Taxation (complicated by the fact that we sell worldwide and support multiple currencies)
  - User support
  - Outreach (pitching events, media, this talk, etc.)
- But it's fun, and you learn a lot!

Summary

Musical modelling with Haskell:

- A model for musical harmony as a Haskell datatype
- Makes use of several advanced functional programming techniques, such as generic programming, GADTs, and type families
- When chords do not fit the model: error correction
- Harmonising melodies
- Generating harmonies
- Recognising harmony from audio sources
- Transporting academic research into industry

Play with it!

chordify®

http://chordify.net
http://hackage.haskell.org/package/HarmTrace
http://hackage.haskell.org/package/FComp