Functional Modelling of Musical Harmony

José Pedro Magalhães

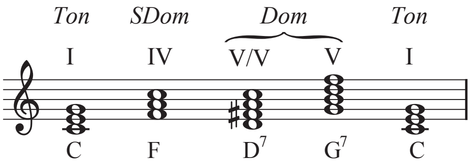
Department of Computer Science, University of Oxford
http://dreixel.net

September 17, 2014
Oxford, United Kingdom

Introduction

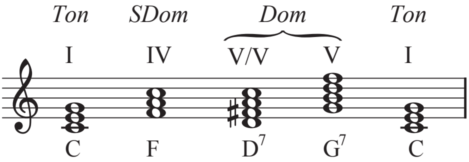
- Modelling musical harmony using Haskell
- Applications of a model of harmony:
  - Musical analysis
  - Finding cover songs
  - Generating chords for melodies
  - Generating chords and melodies
  - Correcting errors in chord extraction from audio sources
  - Chordify—a web-based music player with chord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

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

Simplified harmony theory I

- A *chord* is a group of tones separated by intervals of roughly the same size.
- All music is made out of chords (whether explicitly or not).
- There are 12 different notes. Instead of naming them, we number them relative to the first and most important one, the tonic. So we get I, II$\flat$, II ... VI$\sharp$, VII.
- A chord is built on a root note. So I also stands for the chord built on the first degree, V for the chord built on the fifth degree, etc.
- So the following is a chord sequence: I IV II^7 V^7 I.

Simplified harmony theory II

Models for musical harmony explain the harmonic progression in music:

- Everything works around the *tonic* (I).
- The *dominant* (V) leads to the tonic.
- The *subdominant* (IV) tends to lead to the dominant.
- Therefore, the I IV V I progression is very common.
- There are also *secondary dominants*, which lead to a relative tonic. For instance, II^7 is the secondary dominant of V, and I^7 is the secondary dominant of IV.
- So you can start with I, add one note to get I^7, fall into IV, change two notes to get to II^7, fall into V, and then finally back to I.

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)

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

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

Notes

```haskell
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

$$\textbf{type}\ \mathsf{Melody} = [\mathsf{Note}]$$

$$\mathsf{cMajScale} :: \mathsf{Melody}$$
$$\mathsf{cMajScale} = [\mathsf{c4}, \mathsf{d4}, \mathsf{e4}, \mathsf{f4}, \mathsf{g4}, \mathsf{a4}, \mathsf{b4}]$$

$$\mathsf{cMajScaleRev} :: \mathsf{Melody}$$
$$\mathsf{cMajScaleRev} = \mathsf{reverse}\ \mathsf{cMajScale}$$

$$\mathsf{reverse} :: [\alpha] \to [\alpha]$$
$$\mathsf{reverse}\ [\,] \qquad = [\,]$$
$$\mathsf{reverse}\ (\mathsf{h} : \mathsf{t}) = \mathsf{reverse}\ \mathsf{t} +\!\!+ [\mathsf{h}]$$
$$(+\!\!+) :: [\alpha] \to [\alpha] \to [\alpha]$$
$$(+\!\!+) = \ldots$$

Transposition

Transposing a melody one octave higher:

$$\begin{array}{l}
\mathsf{octaveUp} :: \mathsf{Octave} \to \mathsf{Octave} \\
\mathsf{octaveUp}\ \mathsf{n} = \mathsf{n} + 1 \\[4pt]
\mathsf{noteOctaveUp} :: \mathsf{Note} \to \mathsf{Note} \\
\mathsf{noteOctaveUp}\ (\mathsf{Note}\ \mathsf{r}\ \mathsf{o}) = \mathsf{Note}\ \mathsf{r}\ (\mathsf{octaveUp}\ \mathsf{o}) \\[4pt]
\mathsf{melodyOctaveUp} :: \mathsf{Melody} \to \mathsf{Melody} \\
\mathsf{melodyOctaveUp}\ \mathsf{m} = \mathsf{map}\ \mathsf{noteOctaveUp}\ \mathsf{m}
\end{array}$$

Generation, analysis

Building a repeated melodic phrase:

$$\begin{aligned}&\mathsf{ostinato} :: \mathsf{Melody} \to \mathsf{Melody}\\ &\mathsf{ostinato\ m} = \mathsf{m} \mathbin{+\!\!+} \mathsf{ostinato\ m}\end{aligned}$$

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?

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

$$\text{isCMaj} :: \text{Melody} \rightarrow \text{Bool}$$
$$\text{isCMaj} = (\equiv [\text{A}, \text{B}, \text{C}, \text{D}, \text{E}, \text{F}, \text{G}]) \circ \text{sort} \circ \text{nub} \circ \text{map root}$$

“Details” left out

We have seen only a glimpse of music representation, leaving out:

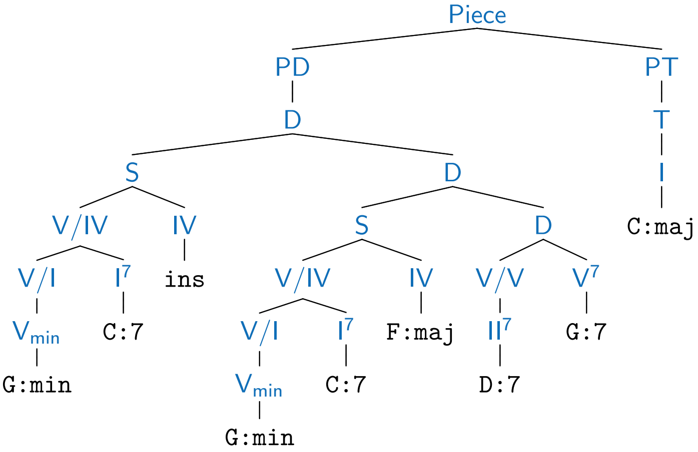
- Rhythm
- Accidentals
- Intervals
- Voicing
- ...

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`

Back to harmony analysis

A hierarchical representation of the harmony of the sequence
G_{min} C^7 G_{min} C^7 F_{Maj} D^7 G^7 C_{Maj}:

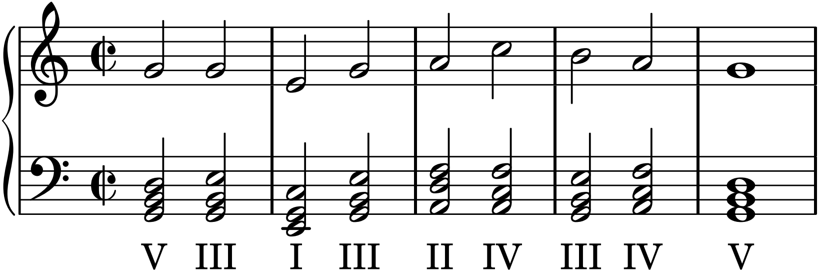


Application: harmonic similarity

- 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

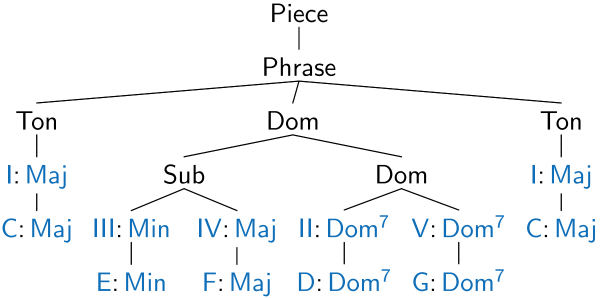
Application: automatic harmonisation of melodies

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



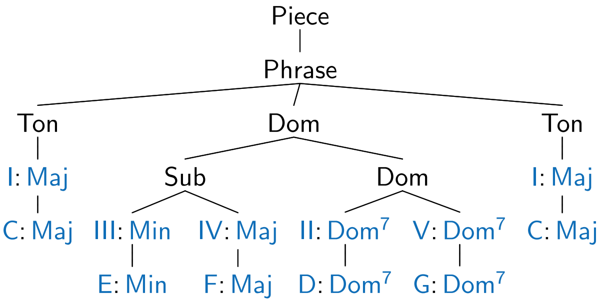
We generate candidate chord sequences, parse them with the harmony model, and select the one with the least errors.

Visualising harmonic structure



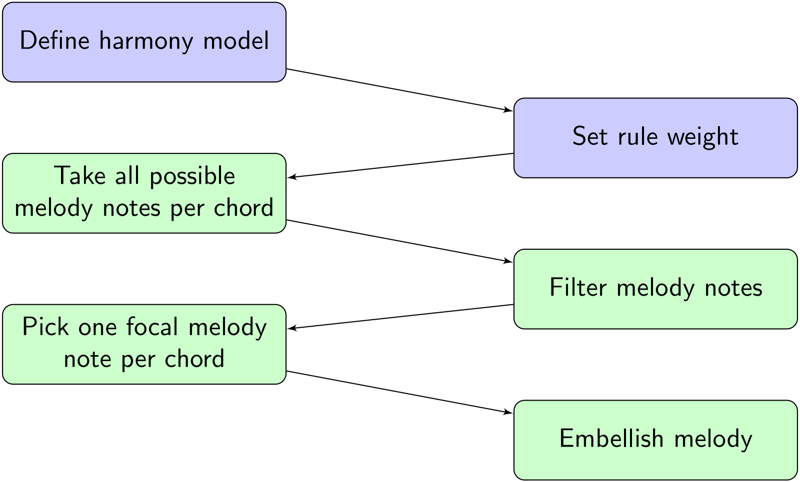
You can see this tree as having been produced by taking the chords in green as input...

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!

System structure



A functional model of harmony

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

A functional model of harmony

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

$$\begin{aligned}\mathsf{Phrase}_{\mathfrak{M}} \rightarrow\; & \mathsf{Ton}_{\mathfrak{M}}\ \mathsf{Dom}_{\mathfrak{M}}\ \mathsf{Ton}_{\mathfrak{M}} \\ |\; & \mathsf{Dom}_{\mathfrak{M}}\ \mathsf{Ton}_{\mathfrak{M}}\end{aligned}$$

A functional model of harmony

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

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

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

A functional model of harmony

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

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

$$\begin{aligned}
\text{Ton}_{\text{Maj}} &\rightarrow \text{I}_{\text{Maj}} \\
\text{Ton}_{\text{Min}} &\rightarrow \text{I}^{m}_{\text{Min}}
\end{aligned}$$

$$\begin{aligned}
\text{Dom}_{\mathfrak{M}} &\rightarrow \text{V}^{7}_{\mathfrak{M}} \\
&\mid\ \text{V}_{\mathfrak{M}} \\
&\mid\ \text{VII}^{0}_{\mathfrak{M}} \\
&\mid\ \text{Sub}_{\mathfrak{M}}\ \text{Dom}_{\mathfrak{M}} \\
&\mid\ \text{II}^{7}_{\mathfrak{M}}\ \text{V}^{7}_{\mathfrak{M}}
\end{aligned}$$

A functional model of harmony

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

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

$$\begin{aligned}
\text{Ton}_{\text{Maj}} &\rightarrow \text{I}_{\text{Maj}} \\
\text{Ton}_{\text{Min}} &\rightarrow \text{I}^{m}_{\text{Min}}
\end{aligned}$$

$$\begin{aligned}
\text{Dom}_{\mathfrak{M}} &\rightarrow \text{V}^{7}_{\mathfrak{M}} \\
&\mid \ \text{V}_{\mathfrak{M}} \\
&\mid \ \text{VII}^{0}_{\mathfrak{M}} \\
&\mid \ \text{Sub}_{\mathfrak{M}}\ \text{Dom}_{\mathfrak{M}} \\
&\mid \ \text{II}^{7}_{\mathfrak{M}}\ \text{V}^{7}_{\mathfrak{M}}
\end{aligned}$$

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

A functional model of harmony

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

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

$$\begin{aligned}
\text{Ton}_{\text{Maj}} &\rightarrow \text{I}_{\text{Maj}} \\
\text{Ton}_{\text{Min}} &\rightarrow \text{I}^{m}_{\text{Min}}
\end{aligned}$$

$$\begin{aligned}
\text{Dom}_{\mathfrak{M}} &\rightarrow \text{V}^{7}_{\mathfrak{M}} \\
&\mid\ \text{V}_{\mathfrak{M}} \\
&\mid\ \text{VII}^{0}_{\mathfrak{M}} \\
&\mid\ \text{Sub}_{\mathfrak{M}}\ \text{Dom}_{\mathfrak{M}} \\
&\mid\ \text{II}^{7}_{\mathfrak{M}}\ \text{V}^{7}_{\mathfrak{M}}
\end{aligned}$$

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

$$\begin{aligned}
\text{I}_{\text{Maj}} &\rightarrow \text{C}: \text{Maj} \\
\text{I}^{m}_{\text{Min}} &\rightarrow \text{C}: \text{Min} \\
\text{V}^{7}_{\mathfrak{M}} &\rightarrow \text{G}: \text{Dom}^{7} \\
\text{VII}^{0}_{\mathfrak{M}} &\rightarrow \text{B}: \text{Dim}
\end{aligned}$$

A functional model of harmony

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

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

$$\begin{aligned}
\mathrm{Ton}_{\mathrm{Maj}} &\to \mathrm{I}_{\mathrm{Maj}} \\
\mathrm{Ton}_{\mathrm{Min}} &\to \mathrm{I}^{m}_{\mathrm{Min}}
\end{aligned}$$

$$\begin{aligned}
\mathrm{Dom}_{\mathfrak{M}} &\to \mathrm{V}^{7}_{\mathfrak{M}} \\
&\mid \mathrm{V}_{\mathfrak{M}} \\
&\mid \mathrm{VII}^{0}_{\mathfrak{M}} \\
&\mid \mathrm{Sub}_{\mathfrak{M}}\ \mathrm{Dom}_{\mathfrak{M}} \\
&\mid \mathrm{II}^{7}_{\mathfrak{M}}\ \mathrm{V}^{7}_{\mathfrak{M}}
\end{aligned}$$

$$\begin{aligned}
\mathrm{Sub}_{\mathrm{Maj}} &\to \mathrm{II}^{m}_{\mathrm{Maj}} \\
&\mid \mathrm{IV}_{\mathrm{Maj}} \\
&\mid \mathrm{III}^{m}_{\mathrm{Maj}}\ \mathrm{IV}_{\mathrm{Maj}} \\
\mathrm{Sub}_{\mathrm{Min}} &\to \mathrm{IV}^{m}_{\mathrm{Min}}
\end{aligned}$$

$$\begin{aligned}
\mathrm{I}_{\mathrm{Maj}} &\to \mathrm{C}: \mathrm{Maj} \\
\mathrm{I}^{m}_{\mathrm{Min}} &\to \mathrm{C}: \mathrm{Min} \\
\mathrm{V}^{7}_{\mathfrak{M}} &\to \mathrm{G}: \mathrm{Dom}^{7} \\
\mathrm{VII}^{0}_{\mathfrak{M}} &\to \mathrm{B}: \mathrm{Dim}
\end{aligned}$$

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

Now in Haskell—I

A GADT encoding musical harmony:

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

$\textbf{data}\ \text{Piece} = \forall \mu :: \text{Mode}.\text{Piece}\ [\,\text{Phrase}\ \mu\,]$

Now in Haskell—I

A GADT encoding musical harmony:

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

$$\textbf{data}\ \mathsf{Piece} = \forall \mu :: \mathsf{Mode}.\mathsf{Piece}\ [\,\mathsf{Phrase}\ \mu\,]$$

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

Now in Haskell—I

A GADT encoding musical harmony:

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

$$\textbf{data}\ \mathsf{Piece} = \forall \mu :: \mathsf{Mode}.\mathsf{Piece}\ [\,\mathsf{Phrase}\ \mu\,]$$

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

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

Now in Haskell—I

A GADT encoding musical harmony:

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

$$\mathbf{data}\ \mathsf{Piece} = \forall \mu :: \mathsf{Mode}.\mathsf{Piece}\ [\,\mathsf{Phrase}\ \mu\,]$$

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

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

$$\begin{array}{l}
\mathbf{data}\ \mathsf{Dom}\ (\mu :: \mathsf{Mode})\ \mathbf{where} \\
\quad \mathsf{Dom}_1 :: \mathsf{SD\ V}\quad \mathsf{Dom}^7 \to \mathsf{Dom}\ \mu \\
\quad \mathsf{Dom}_2 :: \mathsf{SD\ V}\quad \mathsf{Maj} \quad\ \to \mathsf{Dom}\ \mu \\
\quad \mathsf{Dom}_3 :: \mathsf{SD\ VII\ Dim} \quad \to \mathsf{Dom}\ \mu \\
\quad \mathsf{Dom}_4 :: \mathsf{SDom}\ \mu \to \mathsf{Dom}\ \mu \to \mathsf{Dom}\ \mu \\
\quad \mathsf{Dom}_5 :: \mathsf{SD\ II\ Dom}^7 \to \mathsf{SD\ V\ Dom}^7 \to \mathsf{Dom}\ \mu
\end{array}$$

Now in Haskell—II

Scale degrees are the leaves of our hierarchical structure:

$$\begin{array}{l}
\textbf{data}\ \mathsf{DiatonicDegree} = \mathsf{I} \mid \mathsf{II} \mid \mathsf{III} \mid \mathsf{IV} \mid \mathsf{V} \mid \mathsf{VI} \mid \mathsf{VII} \\
\textbf{data}\ \mathsf{Quality} \qquad\quad\; = \mathsf{Maj} \mid \mathsf{Min} \mid \mathsf{Dom}^7 \mid \mathsf{Dim}
\end{array}$$

$$\begin{array}{l}
\textbf{data}\ \mathsf{SD}\ (\delta :: \mathsf{DiatonicDegree})\ (\gamma :: \mathsf{Quality})\ \textbf{where} \\
\quad \mathsf{SurfaceChord} :: \mathsf{ChordDegree} \to \mathsf{SD}\ \delta\ \gamma
\end{array}$$

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 give Arbitrary instances for each of the datatypes in our model.

Generating harmony

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

QuickCheck! We give Arbitrary instances for each of the datatypes in our model.

...but we don't want to do this by hand, for every datatype, and to have to adapt the instances every time we change the model...so we use *generic programming*:

Generating harmony

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

QuickCheck! We give Arbitrary instances for each of the datatypes in our model.

...but we don't want to do this by hand, for every datatype, and to have to adapt the instances every time we change the model...so we use *generic programming*:

$$\begin{aligned} \text{gen} &:: (\text{Representable } \alpha, \text{Generate } (\text{Rep } \alpha)) \\ &\Rightarrow \text{Gen } \alpha \end{aligned}$$

Generating harmony

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

QuickCheck! We give Arbitrary instances for each of the datatypes in our model.

. . . but we don't want to do this by hand, for every datatype, and to have to adapt the instances every time we change the model. . . so we use *generic programming*:

$$\begin{aligned} \mathsf{gen} &:: (\mathsf{Representable}\ \alpha, \mathsf{Generate}\ (\mathsf{Rep}\ \alpha)) \\ &\Rightarrow [\,(\mathsf{String},\mathsf{Int})\,] \rightarrow \mathsf{Gen}\ \alpha \end{aligned}$$

Examples of harmony generation—I

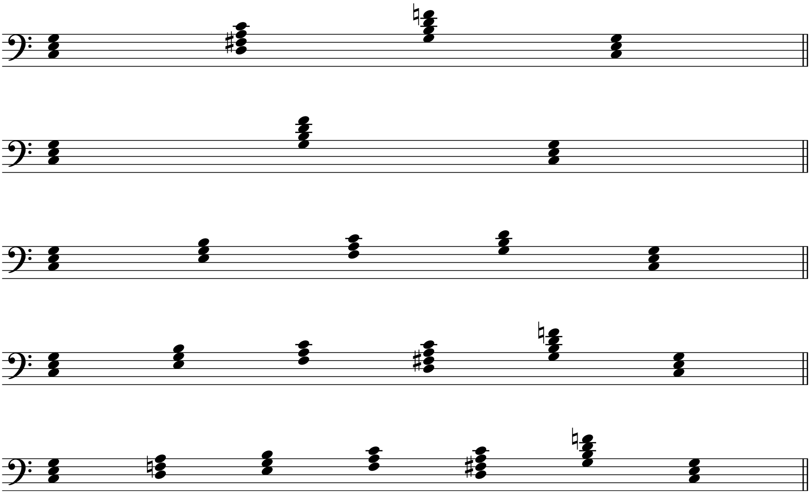
$$\begin{aligned}
&\text{testGen} :: \text{Gen}\ (\text{Phrase}\ \text{Maj}_{\text{Mode}})\\
&\text{testGen} = \text{gen}\ [(\texttt{"Dom4"}, 3), (\texttt{"Dom5"}, 4)]\\
&\text{example} :: \text{IO}\ ()\\
&\text{example} = \textbf{let}\ \text{k} = \text{Key}\ (\text{Note}\ \natural\ \text{C})\ \text{Maj}_{\text{Mode}}\\
&\qquad\qquad\ \textbf{in}\ \text{sample}'\ \text{testGen} \ggg\!= \text{mapM_}\ (\text{printOnKey k})\\
&\text{printOnKey} :: \text{Key} \to \text{Phrase}\ \text{Maj}_{\text{Mode}} \to \text{IO String}
\end{aligned}$$

Examples of harmony generation—I

$$
\begin{aligned}
&\mathsf{testGen} :: \mathsf{Gen}\ (\mathsf{Phrase}\ \mathsf{Maj}_{\mathsf{Mode}}) \\
&\mathsf{testGen} = \mathsf{gen}\ [(\texttt{"Dom4"}, 3), (\texttt{"Dom5"}, 4)] \\
&\mathsf{example} :: \mathsf{IO}\ () \\
&\mathsf{example} = \mathbf{let}\ \mathsf{k} = \mathsf{Key}\ (\mathsf{Note}\ \natural\ \mathsf{C})\ \mathsf{Maj}_{\mathsf{Mode}} \\
&\qquad\qquad\ \mathbf{in}\ \mathsf{sample}'\ \mathsf{testGen} \gg\!= \mathsf{mapM\_}\ (\mathsf{printOnKey}\ \mathsf{k}) \\
&\mathsf{printOnKey} :: \mathsf{Key} \rightarrow \mathsf{Phrase}\ \mathsf{Maj}_{\mathsf{Mode}} \rightarrow \mathsf{IO}\ \mathsf{String}
\end{aligned}
$$

$>$ 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]

Examples of harmony generation—II



Generating a melody for a given harmony

We then generate a melody in 4 steps:

Generating a melody for a given harmony

We then generate a melody in 4 steps:

1. Generate a list of candidate melody notes per chord;

Generating a melody for a given harmony

We then generate a melody in 4 steps:

1. Generate a list of candidate melody notes per chord;
2. Refine the candidates by filtering out obviously bad candidates;

Generating a melody for a given harmony

We then generate a melody in 4 steps:

1. Generate a list of candidate melody notes per chord;
2. Refine the candidates by filtering out obviously bad candidates;
3. Pick one focal candidate melody note per chord;

Generating a melody for a given harmony

We then generate a melody in 4 steps:

1. Generate a list of candidate melody notes per chord;
2. Refine the candidates by filtering out obviously bad candidates;
3. Pick one focal candidate melody note per chord;
4. Embellish the candidate notes to produce a final melody.

Generating a melody for a given harmony

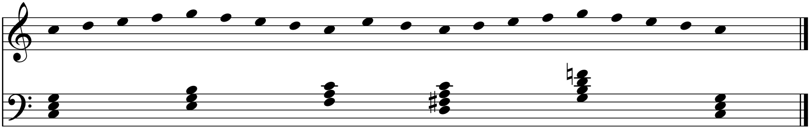
We then generate a melody in 4 steps:

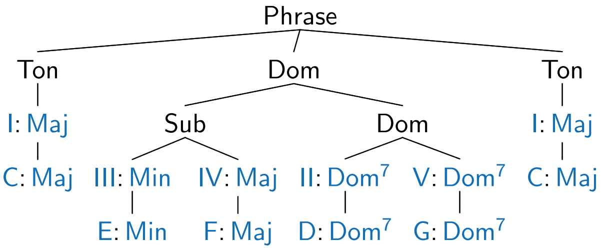
1. Generate a list of candidate melody notes per chord;
2. Refine the candidates by filtering out obviously bad candidates;
3. Pick one focal candidate melody note per chord;
4. Embellish the candidate notes to produce a final melody.

These four steps combine naturally using plain monadic bind:

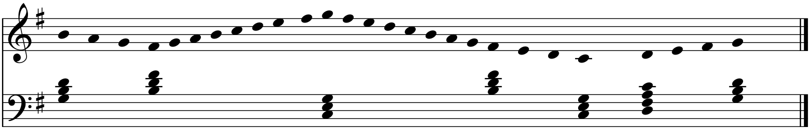
$$
\begin{aligned}
&\mathsf{melody} :: \mathsf{Key} \to \mathsf{State\ MyState\ Song}\\
&\mathsf{melody}\ k \ = \ \mathsf{genCandidates} \gg\!= \mathsf{refine} \gg\!= \mathsf{pickOne} \gg\!= \mathsf{embellish}\\
&\qquad\qquad \gg\!= \mathsf{return} \circ \mathsf{Song}\ k
\end{aligned}
$$

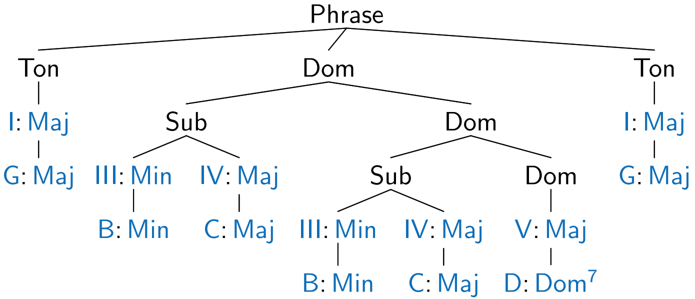
Example I



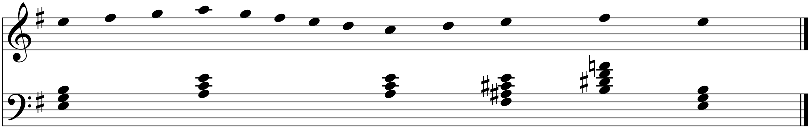

Phrase
Ton
Dom
Ton
I: Maj
Sub
Dom
I: Maj
C: Maj
III: Min
IV: Maj
II: Dom⁷
V: Dom⁷
C: Maj
E: Min
F: Maj
D: Dom⁷
G: Dom⁷

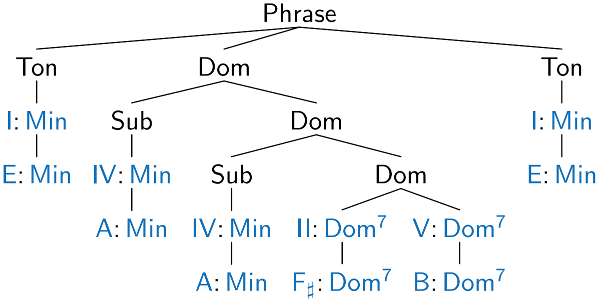
Example II




Phrase
Ton
Dom
Ton
I: Maj
Sub
Dom
I: Maj
G: Maj
III: Min
IV: Maj
Sub
Dom
G: Maj
B: Min
C: Maj
III: Min
IV: Maj
V: Maj
B: Min
C: Maj
D: Dom7

Example III




Phrase
Ton
Dom
Ton
I: Min
Sub
Dom
I: Min
E: Min
IV: Min
Sub
Dom
E: Min
A: Min
IV: Min
II: Dom^7
V: Dom^7
A: Min
F$_\sharp$: Dom^7
B: Dom^7

Another application: 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 Em
		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.

Demo: Chordify

Demo:

chordify®

http://chordify.net

Chordify: architecture

- Frontend
  - Reads user input, such as YouTube/Soundcloud 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 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

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

Play with it!

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