

# Functional Modelling of Musical Harmony

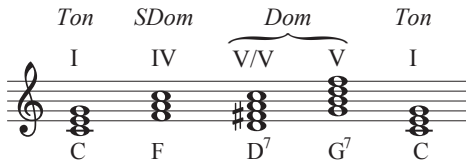
José Pedro Magalhães

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

September 17, 2014  
Oxford, United Kingdom

- ▶ 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

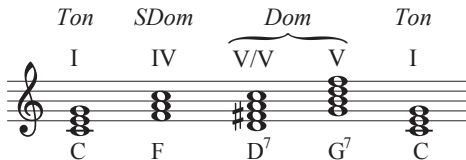
# What is harmony?



The diagram illustrates a sequence of chords on a treble clef staff. Above the staff, the chords are labeled with Roman numerals: I, IV, V/V, V, and I. Above these numerals, the functional categories are labeled: *Ton* above I, *SDom* above IV, *Dom* above V/V and V (with a bracket), and *Ton* above I. Below the staff, the chords are labeled with their letter names: C, F, D<sup>7</sup>, G<sup>7</sup>, and C. The D<sup>7</sup> and G<sup>7</sup> chords are grouped under a bracket labeled *Dom*.

- ▶ 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?



*Ton*    *SDom*    *Dom*    *Ton*

I    IV    V/V    V    I

C    F    D<sup>7</sup>    G<sup>7</sup>    C

- ▶ 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

- ▶ 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 $\sharp$  . . . 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.

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.

**data** Root = A | B | C | D | E | F | G

**type** Octave = Int

**data** Note = Note Root Octave

**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]
```

```
type Melody = [Note]
```

```
cMajScale :: Melody
```

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

```
cMajScaleRev :: Melody
```

```
cMajScaleRev = reverse cMajScale
```

**type** Melody = [Note]

cMajScale :: Melody

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

cMajScaleRev :: Melody

cMajScaleRev = reverse cMajScale

reverse :: [ $\alpha$ ]  $\rightarrow$  [ $\alpha$ ]

reverse [] = []

reverse (h : t) = reverse t ++ [h]

(++) :: [ $\alpha$ ]  $\rightarrow$  [ $\alpha$ ]  $\rightarrow$  [ $\alpha$ ]

(++) = ...

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`

Building a repeated melodic phrase:

ostinato :: Melody  $\rightarrow$  Melody

ostinato m = m  $\ddagger$  ostinato m



Building a repeated melodic phrase:

```
ostinato :: Melody → Melody  
ostinato m = m ++ ostinato m
```

Is a given melody in C major?

```
root :: Note → Root  
root (Note r o) = r  
isCMaj :: Melody → Bool  
isCMaj = (≡ [A, B, C, D, E, F, G]) ∘ sort ∘ nub ∘ 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:

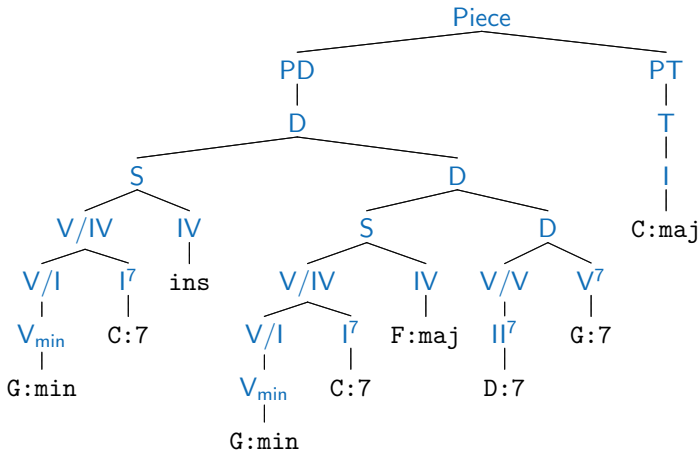
<http://di.uminho.pt/~jno/html/ipm-1011.html>

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  
 $G_{\min}$   $C^7$   $G_{\min}$   $C^7$   $F_{\text{Maj}}$   $D^7$   $G^7$   $C_{\text{Maj}}$ :



- ▶ 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

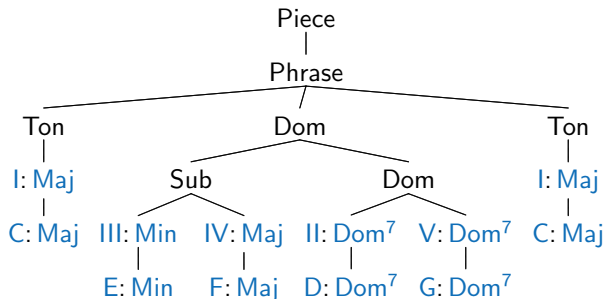
# Application: automatic harmonisation of melodies



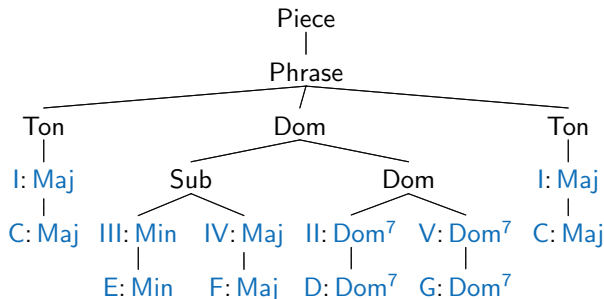
Another practical application of a harmony model is to help selecting good harmonisations (chord sequences) for a given melody:

The image displays a musical score for a single system. The top staff is a treble clef with a common time signature (C). The melody consists of the following notes: G4, A4, B4, A4, G4, F4, E4, D4, C4. The bottom staff is a bass clef with a common time signature (C). The accompaniment consists of the following chords: G major (V), F major (III), E major (I), D major (III), C major (II), B major (IV), A major (III), G major (IV), and F major (V). Below the bass staff, Roman numeral chord symbols are placed under each measure: V, III, I, III, II, IV, III, IV, V.

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

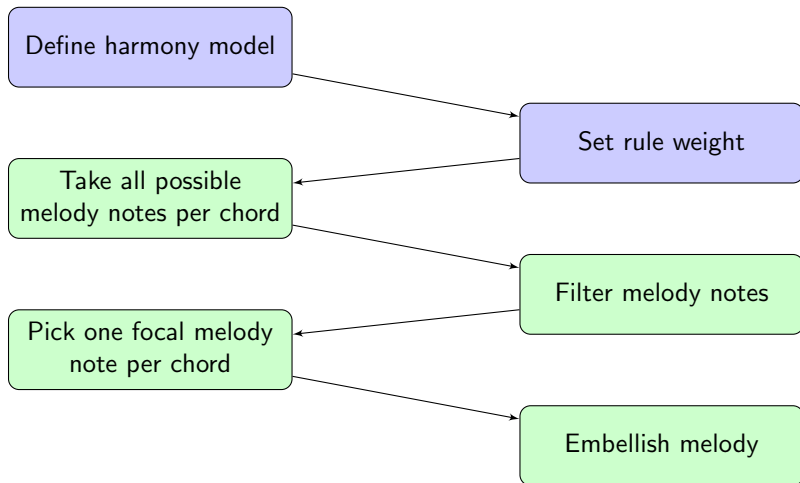


You can see this tree as having been produced by taking the chords in green as input...



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



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

# A functional model of harmony



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

$\text{Phrase}_{\mathfrak{M}} \rightarrow \begin{array}{c} \text{Ton}_{\mathfrak{M}} \text{ Dom}_{\mathfrak{M}} \text{ Ton}_{\mathfrak{M}} \\ | \qquad \text{Dom}_{\mathfrak{M}} \text{ Ton}_{\mathfrak{M}} \end{array}$

# A functional model of harmony



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

$\text{Phrase}_{\mathfrak{M}} \rightarrow \begin{array}{c} \text{Ton}_{\mathfrak{M}} \text{ Dom}_{\mathfrak{M}} \text{ Ton}_{\mathfrak{M}} \\ | \qquad \text{Dom}_{\mathfrak{M}} \text{ Ton}_{\mathfrak{M}} \end{array}$

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

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

# A functional model of harmony



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

$\text{Phrase}_{\mathfrak{M}} \rightarrow \begin{array}{c} \text{Ton}_{\mathfrak{M}} \text{ Dom}_{\mathfrak{M}} \text{ Ton}_{\mathfrak{M}} \\ | \qquad \text{Dom}_{\mathfrak{M}} \text{ Ton}_{\mathfrak{M}} \end{array}$

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

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

$\text{Dom}_{\mathfrak{M}} \rightarrow \begin{array}{c} V_{\mathfrak{M}}^7 \\ | \quad V_{\mathfrak{M}} \\ | \quad VII_{\mathfrak{M}}^0 \\ | \quad \text{Sub}_{\mathfrak{M}} \text{ Dom}_{\mathfrak{M}} \\ | \quad II_{\mathfrak{M}}^7 \quad V_{\mathfrak{M}}^7 \end{array}$

# A functional model of harmony



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

Phrase $_{\mathfrak{M}}$   $\rightarrow$  Ton $_{\mathfrak{M}}$  Dom $_{\mathfrak{M}}$  Ton $_{\mathfrak{M}}$   
 | Dom $_{\mathfrak{M}}$  Ton $_{\mathfrak{M}}$

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

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

Dom $_{\mathfrak{M}}$   $\rightarrow$  V $^7_{\mathfrak{M}}$   
 | V $_{\mathfrak{M}}$   
 | VII $^0_{\mathfrak{M}}$   
 | Sub $_{\mathfrak{M}}$  Dom $_{\mathfrak{M}}$   
 | II $^7_{\mathfrak{M}}$  V $^7_{\mathfrak{M}}$

Sub $_{\text{Maj}}$   $\rightarrow$  II $^m_{\text{Maj}}$   
 | IV $_{\text{Maj}}$   
 | III $^m_{\text{Maj}}$  IV $_{\text{Maj}}$   
 Sub $_{\text{Min}}$   $\rightarrow$  IV $^m_{\text{Min}}$

# A functional model of harmony

Piece<sub>M</sub> → [Phrase<sub>M</sub>]      (M ∈ {Maj, Min})

Phrase<sub>M</sub> → Ton<sub>M</sub> Dom<sub>M</sub> Ton<sub>M</sub>  
                   |            Dom<sub>M</sub> Ton<sub>M</sub>

Ton<sub>Maj</sub> → I<sub>Maj</sub>

Ton<sub>Min</sub> → I<sup>m</sup><sub>Min</sub>

Dom<sub>M</sub> → V<sup>7</sup><sub>M</sub>  
           | V<sub>M</sub>  
           | VII<sup>0</sup><sub>M</sub>  
           | Sub<sub>M</sub> Dom<sub>M</sub>  
           | II<sup>7</sup><sub>M</sub> V<sup>7</sup><sub>M</sub>

Sub<sub>Maj</sub> → II<sup>m</sup><sub>Maj</sub>  
           | IV<sub>Maj</sub>  
           | III<sup>m</sup><sub>Maj</sub> IV<sub>Maj</sub>  
           Sub<sub>Min</sub> → IV<sup>m</sup><sub>Min</sub>

I<sub>Maj</sub> → C: Maj

I<sup>m</sup><sub>Min</sub> → C: Min

V<sup>7</sup><sub>M</sub> → G: Dom<sup>7</sup>

VII<sup>0</sup><sub>M</sub> → B: Dim

# A functional model of harmony

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

Phrase $_{\mathfrak{M}}$   $\rightarrow$  Ton $_{\mathfrak{M}}$  Dom $_{\mathfrak{M}}$  Ton $_{\mathfrak{M}}$   
 | Dom $_{\mathfrak{M}}$  Ton $_{\mathfrak{M}}$

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

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

Dom $_{\mathfrak{M}}$   $\rightarrow$  V $^7_{\mathfrak{M}}$   
 | V $_{\mathfrak{M}}$   
 | VII $^0_{\mathfrak{M}}$   
 | Sub $_{\mathfrak{M}}$  Dom $_{\mathfrak{M}}$   
 | II $^7_{\mathfrak{M}}$  V $^7_{\mathfrak{M}}$

Sub $_{\text{Maj}}$   $\rightarrow$  II $^m_{\text{Maj}}$   
 | IV $_{\text{Maj}}$   
 | III $^m_{\text{Maj}}$  IV $_{\text{Maj}}$   
 Sub $_{\text{Min}}$   $\rightarrow$  IV $^m_{\text{Min}}$

I $_{\text{Maj}}$   $\rightarrow$  C: Maj

I $^m_{\text{Min}}$   $\rightarrow$  C: Min

V $^7_{\mathfrak{M}}$   $\rightarrow$  G: Dom $^7$

VII $^0_{\mathfrak{M}}$   $\rightarrow$  B: Dim

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

# Now in Haskell—I



A GADT encoding musical harmony:

```
data Mode = MajMode | MinMode
```

```
data Piece =  $\forall \mu ::$  Mode.Piece [Phrase  $\mu$ ]
```



# Now in Haskell—I



A GADT encoding musical harmony:

```
data Mode = MajMode | MinMode
```

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

```
data Phrase ( $\mu :: \text{Mode}$ ) where
```

```
PhraseVI :: Ton  $\mu \rightarrow \text{Dom } \mu \rightarrow \text{Ton } \mu \rightarrow \text{Phrase } \mu$ 
```

```
PhraseVI :: Dom  $\mu \rightarrow \text{Ton } \mu \rightarrow \text{Phrase } \mu$ 
```

# Now in Haskell—I



A GADT encoding musical harmony:

```
data Mode = MajMode | MinMode
```

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

```
data Phrase ( $\mu :: \text{Mode}$ ) where
```

```
PhraseVI :: Ton  $\mu \rightarrow \text{Dom } \mu \rightarrow \text{Ton } \mu \rightarrow \text{Phrase } \mu$ 
```

```
PhraseVI :: Dom  $\mu \rightarrow \text{Ton } \mu \rightarrow \text{Phrase } \mu$ 
```

```
data Ton ( $\mu :: \text{Mode}$ ) where
```

```
TonMaj :: SD I Maj  $\rightarrow \text{Ton Maj}_\text{Mode}$ 
```

```
TonMin :: SD I Min  $\rightarrow \text{Ton Min}_\text{Mode}$ 
```

# Now in Haskell—



A GADT encoding musical harmony:

```
data Mode = MajMode | MinMode
```

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

```
data Phrase ( $\mu :: \text{Mode}$ ) where
```

```
  PhraseVI :: Ton  $\mu \rightarrow$  Dom  $\mu \rightarrow$  Ton  $\mu \rightarrow$  Phrase  $\mu$ 
```

```
  PhraseVI ::           Dom  $\mu \rightarrow$  Ton  $\mu \rightarrow$  Phrase  $\mu$ 
```

```
data Ton ( $\mu :: \text{Mode}$ ) where
```

```
  TonMaj :: SD I Maj  $\rightarrow$  Ton MajMode
```

```
  TonMin :: SD I Min  $\rightarrow$  Ton MinMode
```

```
data Dom ( $\mu :: \text{Mode}$ ) where
```

```
  Dom1 :: SD V Dom7  $\rightarrow$  Dom  $\mu$ 
```

```
  Dom2 :: SD V Maj  $\rightarrow$  Dom  $\mu$ 
```

```
  Dom3 :: SD VII Dim  $\rightarrow$  Dom  $\mu$ 
```

```
  Dom4 :: SDom  $\mu \rightarrow$  Dom  $\mu \rightarrow$  Dom  $\mu$ 
```

```
  Dom5 :: SD II Dom7  $\rightarrow$  SD V Dom7  $\rightarrow$  Dom  $\mu$ 
```

Scale degrees are the leaves of our hierarchical structure:

```
data DiatonicDegree = I | II | III | IV | V | VI | VII
```

```
data Quality       = Maj | Min | Dom7 | Dim
```

```
data SD ( $\delta ::$  DiatonicDegree) ( $\gamma ::$  Quality) where  
  SurfaceChord :: ChordDegree  $\rightarrow$  SD  $\delta$   $\gamma$ 
```

# Generating harmony



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

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.

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*:

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*:

```
gen :: (Representable  $\alpha$ , Generate (Rep  $\alpha$ ))  
    => Gen  $\alpha$ 
```



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*:

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

# Examples of harmony generation—I



```
testGen :: Gen (Phrase MajMode)
testGen = gen [("Dom4", 3), ("Dom5", 4)]
example :: IO ()
example = let k = Key (Note ♯ C) MajMode
           in sample' testGen >>= mapM_ (printOnKey k)
printOnKey :: Key → Phrase MajMode → IO String
```

# Examples of harmony generation—I

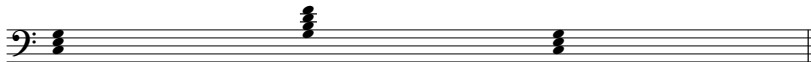


```
testGen :: Gen (Phrase MajMode)
testGen = gen [("Dom4", 3), ("Dom5", 4)]
example :: IO ()
example = let k = Key (Note ♯ C) MajMode
           in sample' testGen >>= mapM_ (printOnKey k)
printOnKey :: Key → Phrase MajMode → IO String
```

> example

```
[C: Maj, D: Dom7, G: Dom7, C: Maj]
[C: Maj, G: Dom7, C: Maj]
[C: Maj, E: Min, F: Maj, G: Maj, C: Maj]
[C: Maj, E: Min, F: Maj, D: Dom7, G: Dom7, C: Maj]
[C: Maj, D: Min, E: Min, F: Maj, D: Dom7, G: Dom7, 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.

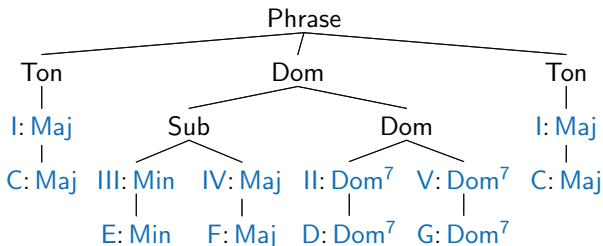
These four steps combine naturally using plain monadic bind:

```
melody :: Key → State MyState Song
melody k = genCandidates >>= refine >>= pickOne >>= embellish
          >>= return ◦ Song k
```

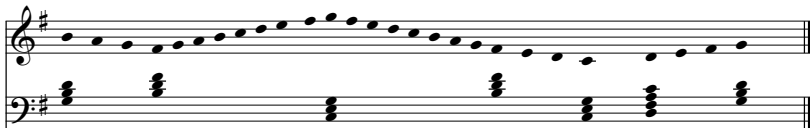
# Example I



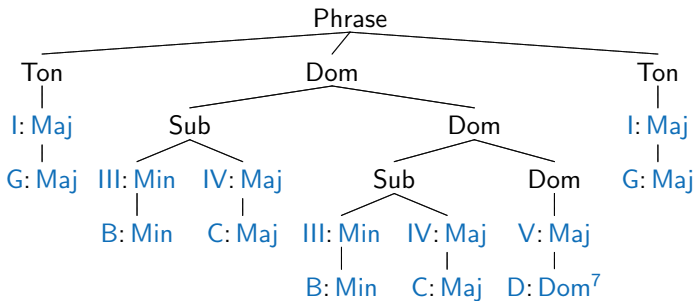
A musical score consisting of two staves. The upper staff is a treble clef with a melodic line of eighth notes. The lower staff is a bass clef with a harmonic accompaniment of chords. The chords are: C major (C4, E4, G4), F major (F4, A4, C5), C major (C4, E4, G4), F major (F4, A4, C5), C major (C4, E4, G4), and C major (C4, E4, G4).



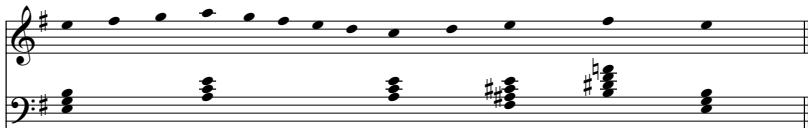
# Example II



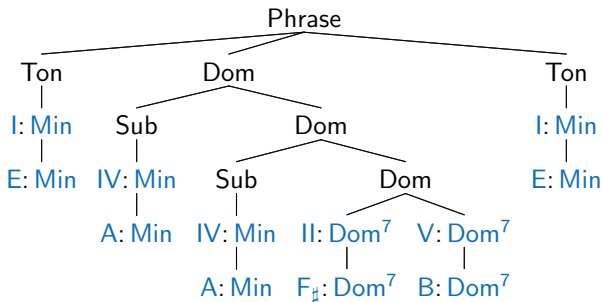
A musical score in G major, consisting of a treble clef staff with a melodic line and a bass clef staff with a harmonic accompaniment. The melody is a simple eighth-note scale: G4, A4, B4, C5, B4, A4, G4. The accompaniment consists of chords: G4 (G-B-D), A4 (A-C-E), B4 (B-D-F#), G4 (G-B-D), F#4 (F#-A-C), E4 (E-G-B), D4 (D-F-A), and G4 (G-B-D).



# Example III



A musical score consisting of two staves. The top staff is in treble clef with a key signature of one sharp (F#). It contains a sequence of eight quarter notes: F#4, G4, A4, B4, C5, B4, A4, G4. The bottom staff is in bass clef with a key signature of one sharp (F#). It contains a sequence of chords: F#2 (F#2, C3, F#3), G2 (G2, C3, F#3), A2 (A2, C3, F#3), B2 (B2, C3, F#3), C3 (C3, F#3, G3), B2 (B2, C3, F#3), A2 (A2, C3, F#3), and G2 (G2, C3, F#3).



# 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<sup>®</sup>

<http://chordify.net>

- ▶ 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



- ▶ 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

## 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`