Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

Functional Modeling of Musical Harmony

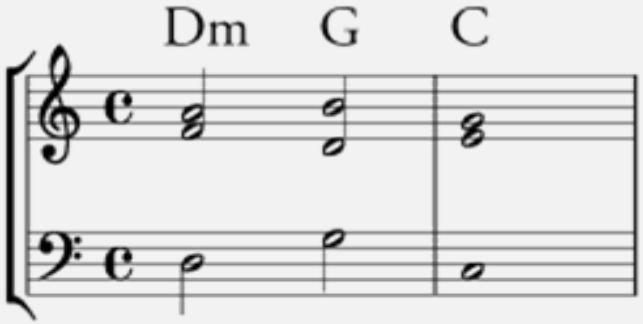
José Pedro Magalhães
joint work with Bas de Haas

Dept. of Information and Computing Sciences, Utrecht University
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
Web pages: `http://www.cs.uu.nl/wiki/Center`

January 7, 2011

Outline

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

A crash course on harmony theory I

A chord can be viewed as a group of tones that are separated by intervals of roughly the same size. Three factors are very important in how a chord is perceived:

A crash course on harmony theory I

A chord can be viewed as a group of tones that are separated by intervals of roughly the same size. Three factors are very important in how a chord is perceived:

- The intervals used in a chord, e.g. minor and major.
- The tone on which the chord has been built, called the root note.
- The surrounding chord sequence in which the chord occurs.

A crash course on harmony theory II

Models for musical harmony explain the harmonic progression in music:

- Based on the interval structure of a chord, we assign a class to a chord: Major, Minor, Dominant, ...
- Every tone in an octave can be used as a root note yielding scale degrees. These are typically numbered in Roman letters: I, II♭, II ... VI♯, VII.
- The function of a chord is determined by the surrounding chords in the sequence.

A crash course on harmony theory II

Models for musical harmony explain the harmonic progression in music:

- Based on the interval structure of a chord, we assign a class to a chord: Major, Minor, Dominant, ...
- Every tone in an octave can be used as a root note yielding scale degrees. These are typically numbered in Roman letters: I, II♭, II ... VI♯, VII.
- The function of a chord is determined by the surrounding chords in the sequence.

(Caveat: for simplicity we abstract from key and work in C major only. Therefore I corresponds to C, II to D, etc.)

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)

Representing harmony as a datatype

```
data Piece = Piece [Phrase]

data Phrase = P1 Ton Dom
            | P2 Ton

data Dom = D1 SDom Dom
         | D2 (Deg V DomCl)
         | D3 (Deg V MajCl)
         | ...

data Ton = T1 (Deg I MajCl)
         | T2 (Deg I MajCl) (Deg IV MajCl) (Deg I MajCl)
         | ...

data SDom = S1 (Deg II MinCl)
          | S2 (Deg IV MajCl)
          | ...
```

Handling degrees

First, we need degrees and chord classes at the type level:

```
data I; data II; ... data VII;
data Maj_Cl; data Min_Cl; data Dom_Cl; ...
```

We use them as phantom type parameters to Deg:

```
data Deg δ γ = Deg {chordRole  :: Int
                   , chordClass :: Class
                   , chordText  :: String}
data Class = Maj_Cl | Min_Cl | Dom_Cl ...
```

Parsing chord sequences

We now want to parse (textual) chord sequences into our datatype representing musical harmony:

```
data ParserMusic α    -- abstract
class Parse α where
  parse :: ParserMusic α
```

Parsing chord sequences

We now want to parse (textual) chord sequences into our datatype representing musical harmony:

```
data ParserMusic α    -- abstract
class Parse α where
  parse :: ParserMusic α
```

Most instances are trivial:

```
instance Parse Phrase where
  parse  =  P1 <$> parse <*> parse
        <|> P2 <$> parse
```

Parsing chord sequences

We now want to parse (textual) chord sequences into our datatype representing musical harmony:

```
data ParserMusic α    -- abstract
class Parse α where
  parse :: ParserMusic α
```

Most instances are trivial:

```
instance Parse Phrase where
  parse  =  P1 <$> parse <*> parse
        <|> P2 <$> parse
```

(So trivial that we do not write them; we use a generic parser.)

Parsing degrees

The interesting case is that for degrees:

```
instance (ToDegree δ, ToClass γ) ⇒ Parse (Deg δ γ) where
  parse = f <$> pChord deg clss where
    f (a, b, c) = Deg a b c
    deg = toDegree (⊥ :: δ)
    clss = toClass   (⊥ :: γ)
```

Parsing degrees

The interesting case is that for degrees:

```
instance (ToDegree δ, ToClass γ) ⇒ Parse (Deg δ γ) where
  parse = f <$> pChord deg clss where
    f (a, b, c) = Deg a b c
    deg = toDegree (⊥ :: δ)
    clss = toClass   (⊥ :: γ)

class ToDegree δ where toDegree :: δ → Int
class ToClass  γ where toClass  :: γ → Class

instance ToDegree I       ...   -- instances for all degrees
instance ToClass  Maj_CI  ...   -- instances for all classes

pChord :: Int → Class → ParserMusic (Int, Class, String)
pChord = ...
```

Increasing complexity

The model we presented so far is very simple and cannot account for:

Repeated chords $\underline{G_{Maj}\ G^7}\ C_{Maj}$

Increasing complexity

The model we presented so far is very simple and cannot account for:

Repeated chords $\underline{G_{Maj}\ G^7}\ C_{Maj}$

Diatonic secondary dominants $\underline{E_{min}\ A_{min}\ D_{min}}\ G^7\ C_{Maj}$

Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

Increasing complexity

The model we presented so far is very simple and cannot account for:

Repeated chords $\underline{G_{Maj}\ G^7}\ C_{Maj}$

Diatonic secondary dominants $\underline{E_{min}\ A_{min}\ D_{min}}\ G^7\ C_{Maj}$

Chromatic secondary dominants $\underline{D^7}\ G^7\ C_{Maj}$

Increasing complexity

The model we presented so far is very simple and cannot account for:

Repeated chords $\underline{G_{Maj}\ G^7}\ C_{Maj}$

Diatonic secondary dominants $\underline{E_{min}\ A_{min}\ D_{min}}\ G^7\ C_{Maj}$

Chromatic secondary dominants $\underline{D^7}\ G^7\ C_{Maj}$

Minor key dominant borrowing $\underline{G_{min}}\ C_{Maj}$

Increasing complexity

The model we presented so far is very simple and cannot account for:

Repeated chords $\underline{G_{Maj}\ G^7}\ C_{Maj}$

Diatonic secondary dominants $\underline{E_{min}\ A_{min}\ D_{min}}\ G^7\ C_{Maj}$

Chromatic secondary dominants $\underline{D^7}\ G^7\ C_{Maj}$

Minor key dominant borrowing $\underline{G_{min}}\ C_{Maj}$

Tritone substitutions $\underline{A\flat^7}\ G^7\ C_{Maj}$

...

Handling repeated chords

Easy: adapt the Deg type and pChord:

```
data Deg δ γ = Deg {chordRole  :: Int
                   , chordClass :: Class
                   , chordText  :: [String]}
pChord :: Int → Class → ParserMusic (Int, Class, [String])
pChord = ...    -- adapted accordingly
```

Secondary dominants—datatype representation

Here the fun begins! Adapt the Deg type again:

$$\textbf{type}\ \mathsf{Deg}_{\mathsf{SD}}\ \ \delta\ \gamma = \mathsf{BDeg}_{\mathsf{SD}}\ \delta\ \gamma$$

$$\begin{aligned}&\textbf{data}\ \mathsf{BDeg}_{\mathsf{SD}}\ \delta\ \gamma\ \textbf{where}\\ &\quad \mathsf{Cons}\ ::\ \mathsf{BDeg}_{\mathsf{SD}}\ (\mathsf{Perf}_{\mathsf{V}}\ \delta)\ \mathsf{Dom}_{\mathsf{Cl}}\\ &\qquad\quad \rightarrow \mathsf{Deg}\ \delta\ \gamma \rightarrow \mathsf{BDeg}_{\mathsf{SD}}\ \delta\ \gamma\\ &\quad \mathsf{Base}\ ::\ \mathsf{Deg}\ \delta\ \gamma \rightarrow \mathsf{BDeg}_{\mathsf{SD}}\ \delta\ \gamma\end{aligned}$$

Secondary dominants—datatype representation

Here the fun begins! Adapt the Deg type again:

```
type Deg_SD δ γ = BDeg_SD δ γ

data BDeg_SD δ γ where
  Cons :: BDeg_SD (Perf_V δ) Dom_CI
       → Deg δ γ → BDeg_SD δ γ
  Base :: Deg δ γ → BDeg_SD δ γ

data Deg δ γ = Deg ...    -- as before

type family  Perf_V δ :: ⋆   -- computes the perfect fifth
type instance Perf_V I  = V
type instance Perf_V V = II
...
```

Secondary dominants—datatype representation

Here the fun begins! Adapt the Deg type again:

```
type Deg_SD δ γ = BDeg_SD δ γ (Su (Su (Su Ze)))
data BDeg_SD δ γ η where
  Cons :: BDeg_SD (Perf_V δ) Dom_CI η
       → Deg δ γ → BDeg_SD δ γ (Su η)
  Base :: Deg δ γ → BDeg_SD δ γ (Su η)

data Deg δ γ = Deg ...    -- as before

type family   Perf_V δ :: ⋆   -- computes the perfect fifth
type instance Perf_V I  = V
type instance Perf_V V  = II
...

data Ze      -- type level zero
data Su η    -- type level successor
```

Secondary dominants—instances

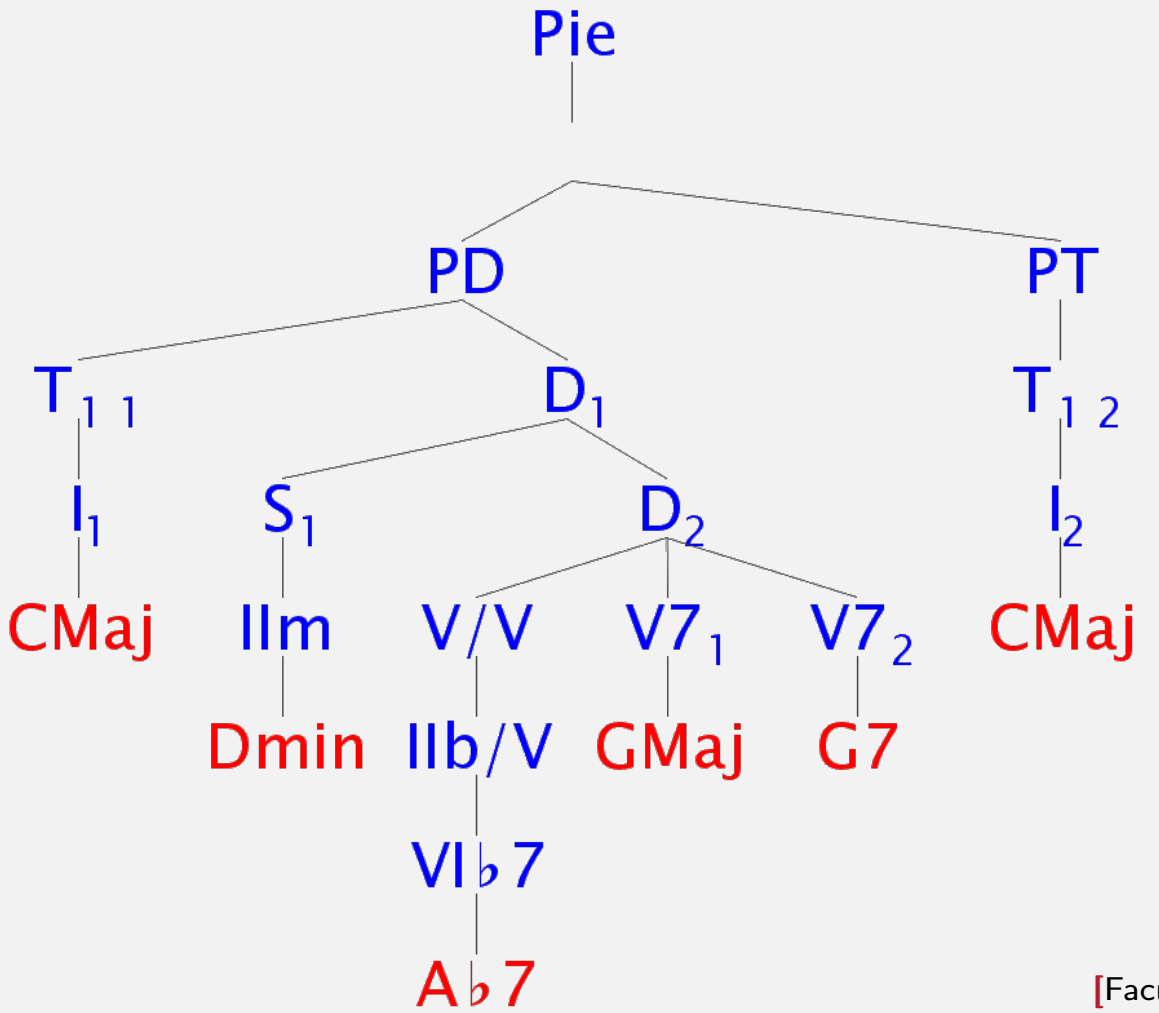
We also need an instance of Parse for $\mathsf{BDeg}_{\mathsf{SD}}$:

```
instance  (  Parse (Deg δ γ)
          ,  Parse (BDeg_SD (Perf_V δ) Dom_CI η      ))
          ⇒  Parse (BDeg_SD δ          γ      (Su η)) where
  parse  =   Base <$> parse
         <|> Cons <$> parse <*> parse
```

The code for parse is trivial, but the instance head is not.

Example

Parsing the sequence C_{Maj} D_{min} $A\flat^7$ G_{Maj} G^7 C_{Maj}:



Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

Summary

What we have so far:

- A model for musical harmony as a Haskell GADT
- The datatypes can change, the code does not have to:
  - Generic parser
  - Generic pretty-printer
  - Generic enumerator (automatically generate valid harmonic sequences)
- When chords do not fit the model: error correction

What we are working on:

- A similarity measure for our musical structures (or a generic diff?)

Limitations

- Performance:
  - Type checking: exponential on η
  - At runtime: grammar ambiguity
- Model:
  - Deal with minor keys
  - Modulations