Functional Modelling of Musical Harmony

and its applications

José Pedro Magalhães

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

October 22, 2012
ICT.OPEN, Rotterdam, The Netherlands

This talk

- Modelling musical harmony using Haskell

This talk

- Modelling musical harmony using Haskell
- Avoiding programming

This talk

UNIVERSITY OF OXFORD

- Modelling musical harmony using Haskell
- Avoiding programming
- Applications of the harmony model:

This talk

- Modelling musical harmony using Haskell
- Avoiding programming
- Applications of the harmony model:
  - Musical analysis

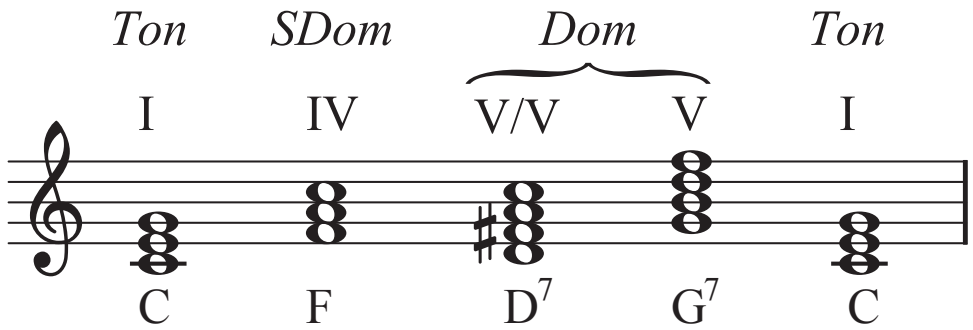
This talk

- Modelling musical harmony using Haskell
- Avoiding programming
- Applications of the harmony model:
  - Musical analysis
  - Finding cover songs

This talk

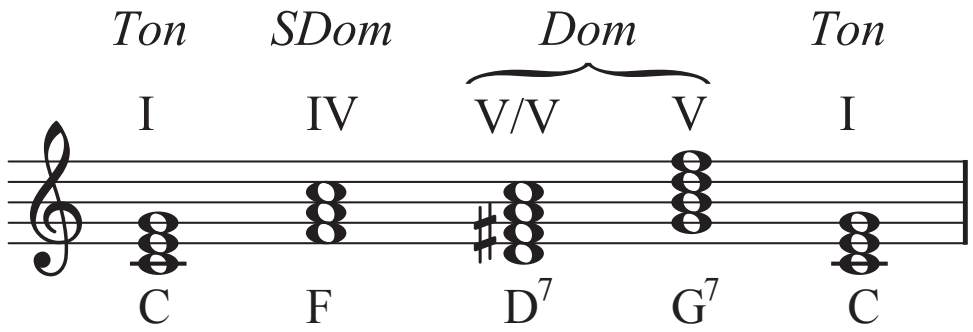
- Modelling musical harmony using Haskell
- Avoiding programming
- Applications of the harmony model:
  - Musical analysis
  - Finding cover songs
  - Correcting errors in chord extraction from audio sources

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

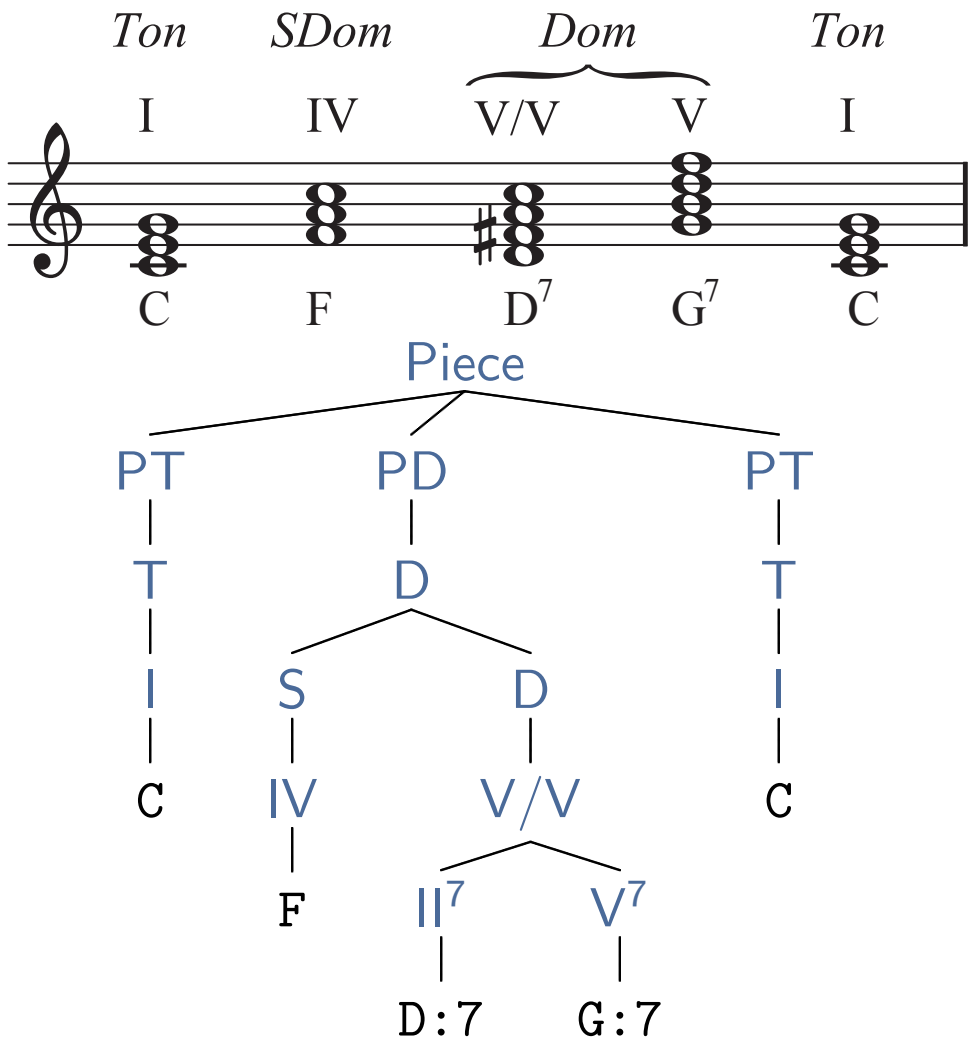
What is harmony?



- 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)

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.

Representing harmony as a datatype, naively

```
data Piece    = Piece  [Phrase]
data Phrase   = PT     Ton      | PD     Dom
data Ton      = T_IMaj DegNum
data Dom      = D_VMaj DegNum   | D_SDom SDom Dom
data SDom     = S_IVMaj DegNum
data DegNum   = I | II | III ...
```

Representing harmony as a datatype, naively

```
data Piece    = Piece  [Phrase]
data Phrase   = PT     Ton      | PD     Dom
data Ton      = T_IMaj DegNum
data Dom      = D_VMaj DegNum   | D_SDom SDom Dom
data SDom     = S_IVMaj DegNum
data DegNum   = I | II | III ...
```

Problem: the term Piece [PT (T_{IMaj} II)] typechecks, but represents an invalid harmonic structure. T_{IMaj} should only take I as argument. How can we enforce this? (And why do we want it?)

Representing harmony as a datatype I

```
data Piece  = Piece  [Phrase]
data Phrase = PT     Ton      | PD Dom
data Ton    = T_IMaj  (Deg I)
data Dom    = D_VMaj  (Deg V) | D_SDom SDom Dom
data SDom   = S_IVMaj (Deg IV)

data Deg δ = Deg DegNum
```

We do not export the Deg constructor. So how do we build Degs?

Representing harmony as a datatype II

```
deg :: ToDegree δ ⇒ δ → Deg δ
deg d = Deg (toDegree d)
```

We need degrees at the type level:

```
data I; data II; ... data VII;
```

And type-to-value conversions for degrees:

```
class     ToDegree δ where toDegree :: δ → DegNum
instance ToDegree I   where toDegree _ = I
instance ToDegree II  where toDegree _ = II
...    -- instances for all degrees
```

Representing harmony as a datatype III

Now the term Piece [PT (T_{IMaj} (deg ($\perp$ :: II)))] does not typecheck, as we wanted:

```
*ghci> Piece [PT (TIMaj (deg (undefined :: II)))]
    Couldn't match expected type 'I'
      with actual type 'II'
```

Representing harmony as a datatype IV

We have a well-typed harmony model in Haskell that accepts only "correct" harmonic sequences.

Here we have only seen a very basic subset of our model. In reality, it consists of 14 datatypes with a total of 46 constructors.

Also, while developing the model we have to change it often, trying to find a balance between complexity and expressiveness.

Parsing chord sequences I

We now consider the problem of parsing (textual) chord sequences into our datatype representing musical harmony:

```
class Parse α where
  parse :: ParserMusic α
```

Parsing chord sequences II

Most instances are trivial:

```
instance Parse Phrase where
  parse  =  PT <$> parse
        <|> PD <$> parse
instance Parse Ton where
  parse  =  T_IMaj  <$> parse
instance Parse Dom where
  parse  =  D_VMaj  <$> parse
        <|> D_SDom <$> parse <*> parse
...
```

Parsing chord sequences II

Most instances are trivial:

```
instance Parse Phrase where
  parse  =  PT <$> parse
        <|> PD <$> parse
instance Parse Ton where
  parse  =  T_IMaj  <$> parse
instance Parse Dom where
  parse  =  D_VMaj  <$> parse
        <|> D_SDom <$> parse <*> parse
...
```

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

Parsing degrees

For degrees we need an adhoc parser:

```
instance (ToDegree δ) ⇒ Parse (Deg δ) where
  parse = pChord (toDegree (⊥ :: δ))

pChord :: DegNum → ParserMusic (Deg δ)
pChord = ...
```

Again we use the type-to-value conversion class ToDegree.

Increasing complexity

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

Repeated chords $\underline{G_{Maj}\ G_{Maj}}\ C_{Maj}$

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

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

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 = Deg {chordRole :: DegNum
                 , numReps  :: Int}

pChord :: DegNum → ParserMusic (Deg δ)
pChord = ...    -- adapted accordingly
```

Secondary dominants—datatype representation

Here the fun begins! We adapt the Deg type again:

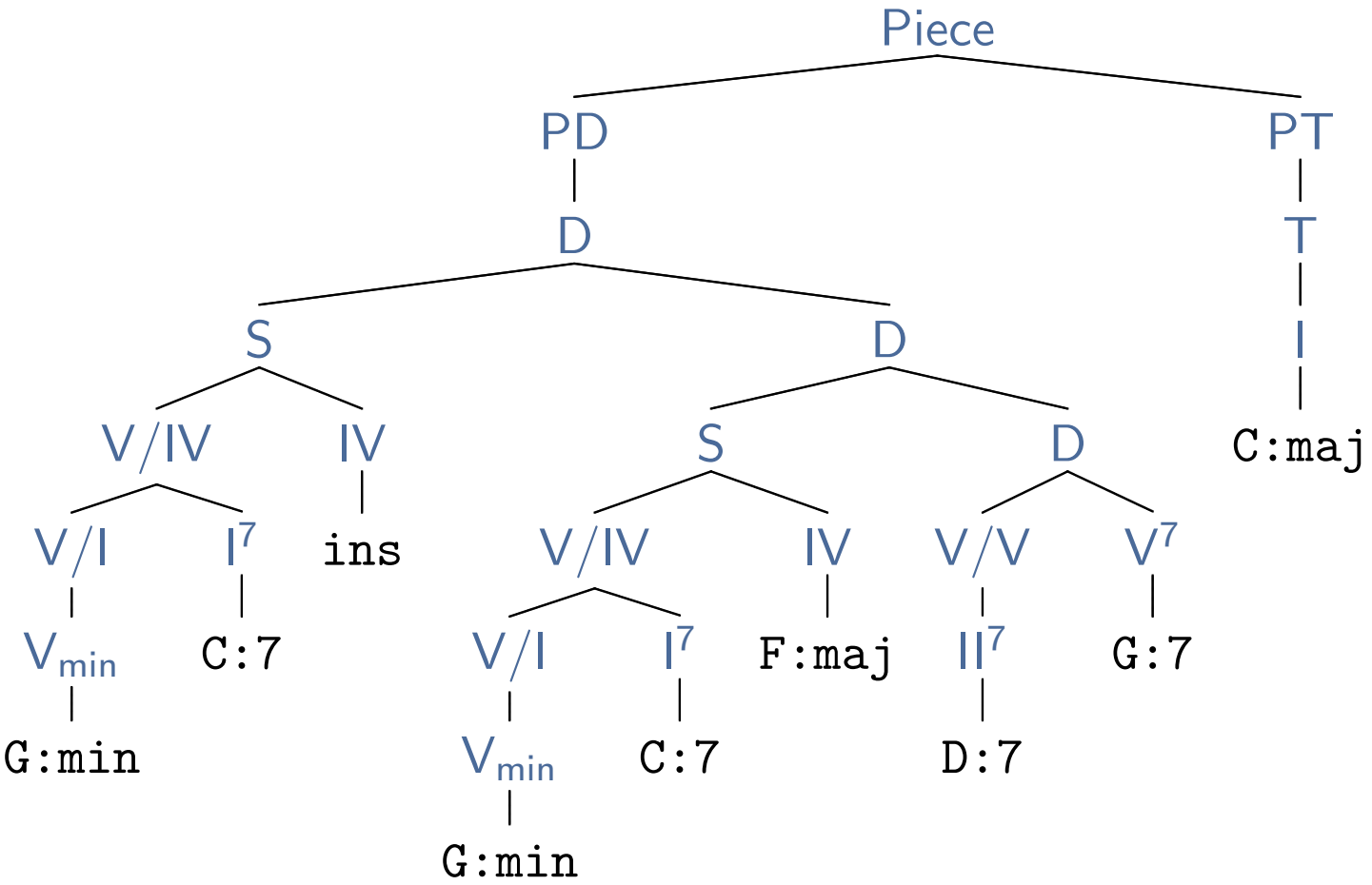
```
type Deg_SD δ = BDeg_SD δ
data BDeg_SD δ where
  Cons :: BDeg_SD (Perf_V δ) → Deg δ → BDeg_SD δ
  Base ::                      Deg δ → BDeg_SD δ

data Deg δ = Deg ...    -- as before

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

Application: harmony analysis

Parsing 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: chord recognition

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.

Application: chord recognition

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: `http://chordify.net`

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 diff
- Multiple models, lots of code reuse
- When chords do not fit the model: error correction
- Recognising harmony from audio sources

Play with it!

`http://hackage.haskell.org/package/HarmTrace`

`http://chordify.net`