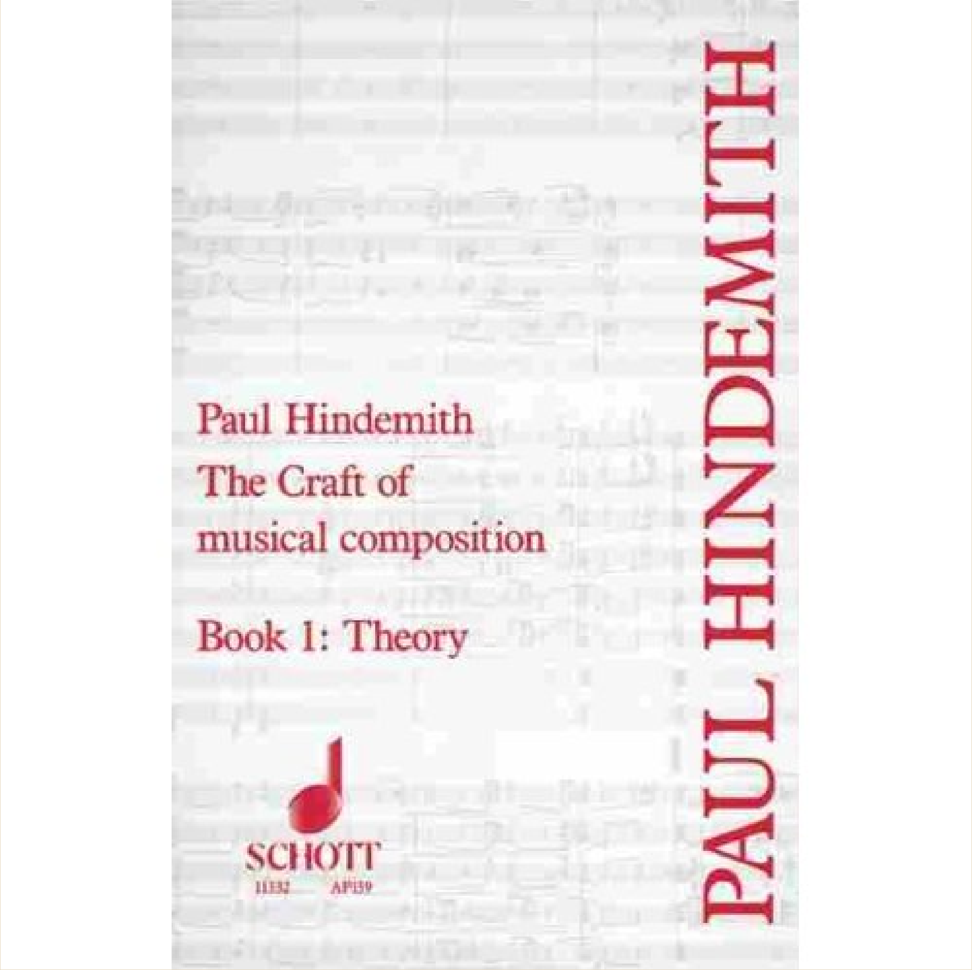
Hindemith

in Haskell (II)



Paul Hindemith
The Craft of
musical composition

Book 1: Theory

SCHOTT
11332 AP139

PAUL HINDEMITH

Fundamentals of Music

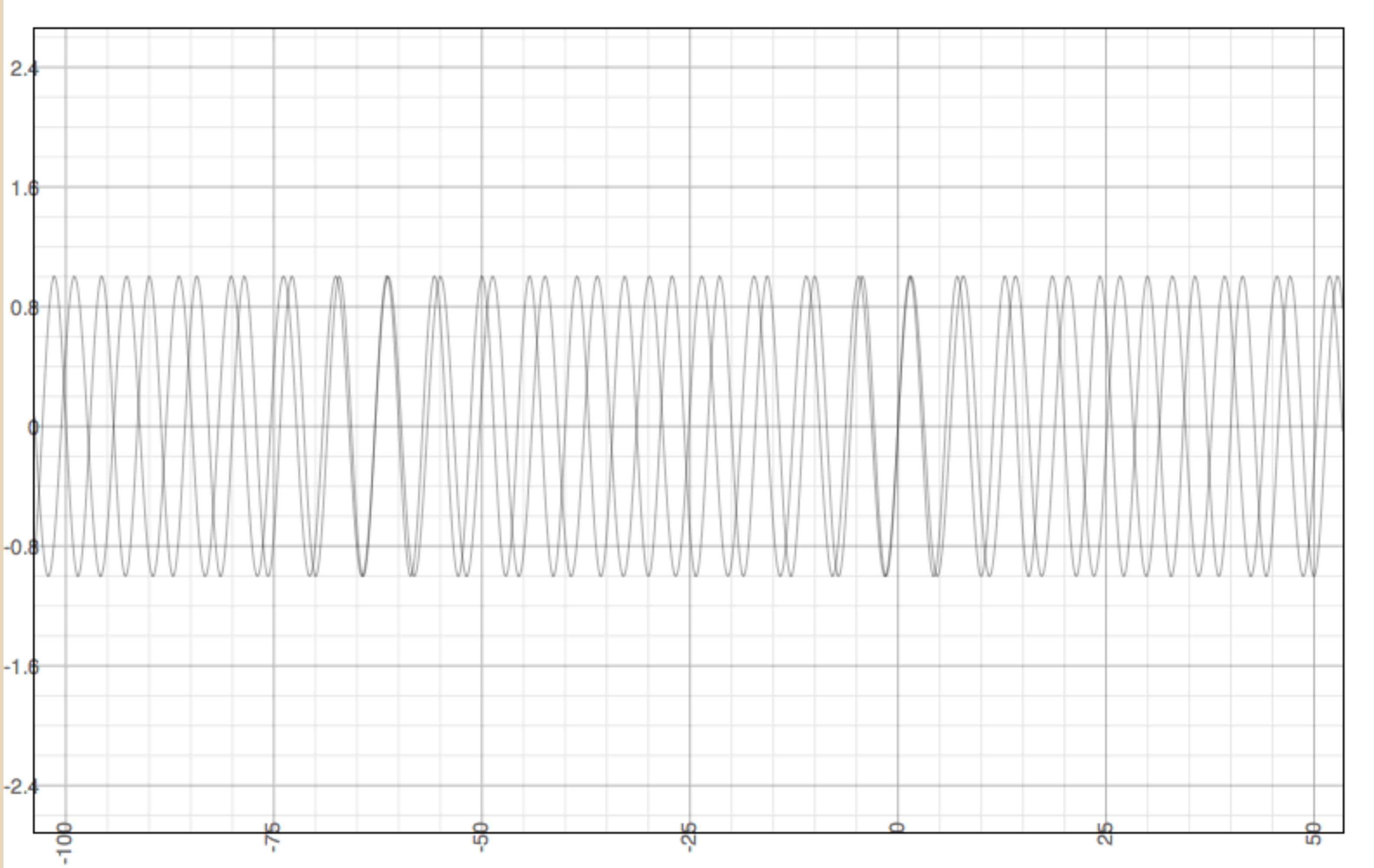
- Notes
  - Melodies

- Intervals
  - Harmonies

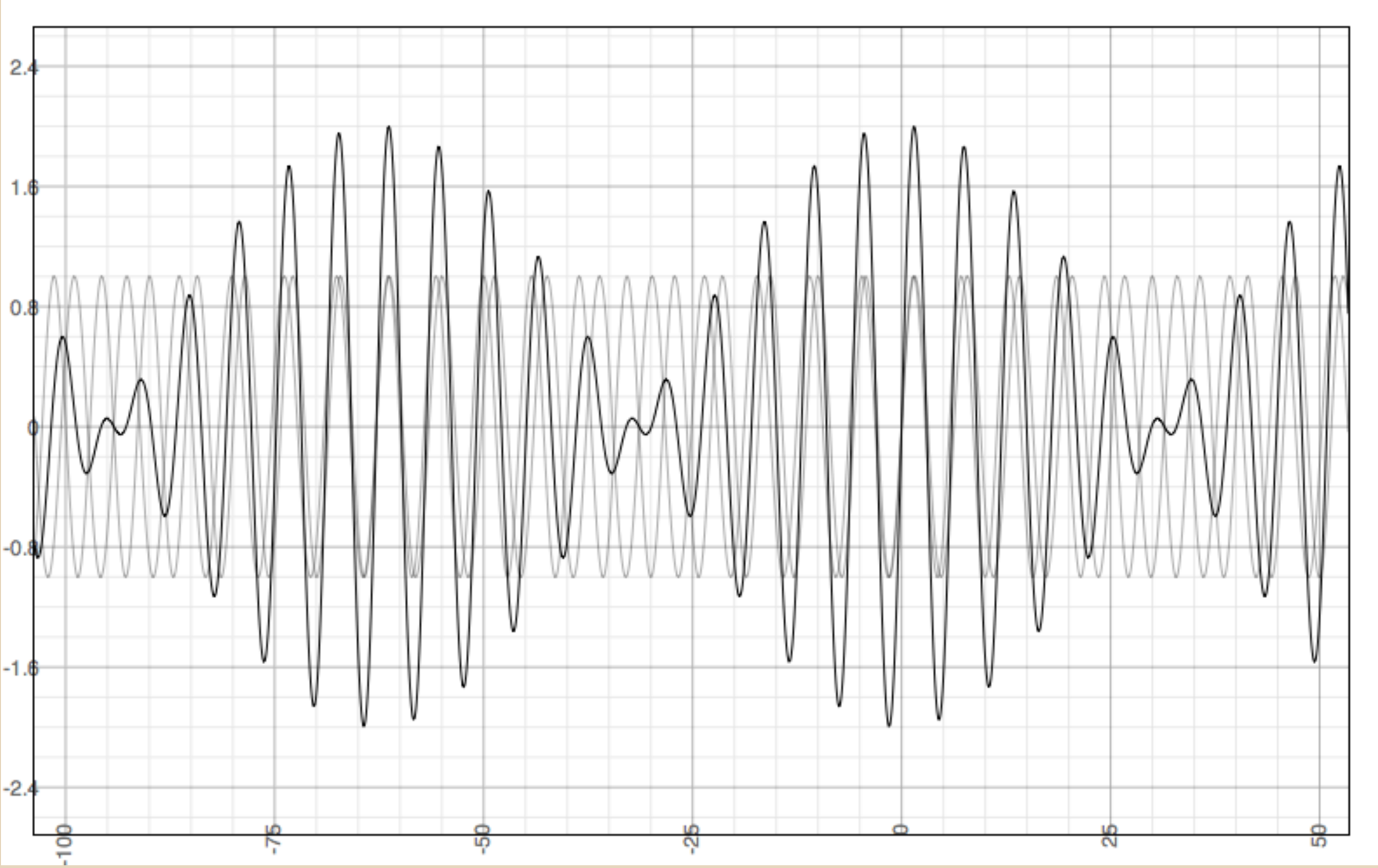
- Chords
  - Progressions

Fundamentals of Music

- Sound is waves in air
  - notes have characteristic frequencies
- Frequency doubling is special
  - the "octave"
- Notes playing together generate interference
- Musical instruments aren't perfect
  - each note has "overtones"



2.4
1.6
0.8
0
-0.8
-1.6
-2.4
-100
-75
-50
-25
0
25
50


2.4
1.6
0.8
0
-0.8
-1.6
-2.4
-100
-75
-50
-25
0
25
50

Hindemith's Scale

Constrained roots of overtones of the base note:

- G (3/2), F (4/3), A (5/3), E (5/4), E♭ (6/5)
- Ab (8/5)

Constrained roots of overtones of these notes:

- D (9/8), B♭ (16/9), D♭ (16/15), B (15/8)

Derivations of the tritone from these notes:

- G♭ (45/32)

Hindemith's Scale

Result: G, F, A, E, E♭, A♭, D, B♭, D♭, B, G♭

```
scale = c:db:d:eb:e:f:fs:g:ab:a:bb:b:[]
```

Normalising Notes

```
normalise' :: (Note a, Ord a) => a -> a -> (a, Int)
normalise' base tone = n' base tone 0
  where
    n' base tone o =  if tone >= octave base
                      then n' base (tone `undertone` 2) (o +
1)
                      else (
                        if tone < base
                        then n' base (octave tone) (o - 1)
                        else (tone, o))

normalise base tone = fst $ normalise' base tone
```

Naming Notes

```
data NamedNote  = C | Db | D | Eb | E | F | Fs | Gb | G | Ab | A | Bb | B
    | Octave Int NamedNote | Sharp NamedNote Double
    | Flat NamedNote Double | Unknown Int Int
        deriving (Eq, Show, Ord)

notes = [(c, C), (db, Db), (d, D), (eb, Eb), (e, E), (f, F), (fs, Fs), (gb, Gb), (g, G),
(ab, Ab), (a, A), (bb, Bb), (b, B)]

toNamedNote note = denormalise octaves . toName . best $ diffs
  where
    (normNote, octaves) = normalise' (fst . head $ notes) note
    denormalise 0 note = note
    denormalise n note = Octave n note
    diffs = map (\(value, name) -> (pitch normNote - pitch value, name)) notes
    best = minimumBy (\(a,n) (b,n') -> compare (abs a) (abs b))
    toName (0.0, name) = name
    toName (x, name) | x > 0 = Sharp name (x / pitch note)
    toName (x, name) = Flat name (x / pitch note)
```

Intervals

- Interference is important

- two notes of frequency 'a' and 'b' generate:
  - diff(a, b) = b - a
  - diff(b - a, a) = b - 2a or 2a - b
  - diff(b - a, b) = b + a

- e.g. C & D (= 9/8 C)
  - 1/8 C, 7/8 C, 9/8 C

Intervals

```
intervalNotes a b = drop 2 $ nub [a, b, c, d, e]
  where
    c = pitchDiff a b
    d = pitchDiff a c
    e = pitchDiff b c

pitchDiff a b = if p1 == p2 then fromRatioTuple result
                            else error "mismatched base tones"
  where
    (p1, o1, r1) = toRatioTuple a
    (p2, o2, r2) = toRatioTuple b
    result = if (o1, r1) == (o2, r2)
             then (p1, o1, r1)
             else (p1, numerator ratio, denominator ratio)
    num = abs $ o1 * r2 - o2 * r1
    denom = r1 * r2
    ratio = num % denom
```

"Quality" of an Interval

```
> map toNamedNote $ intervalNotes c g
[Octave (-1) C]
```

```
>map toNamedNote $ intervalNotes c db
[Octave (-3) C, Octave (-1) (Flat B -8.93e-
3)]
```

Quality Measures

- How many distinct tones?
- are the generated tones Octave doublings of existing tones?
- What's the largest departure from a whole note?
- How many new (normalised) notes are introduced?
- How far down the tone progression is the top note from the root?

Quality Measures

- How many distinct tones?

```
length . nub . map (normaliseNote . toNamedNote) $ a:b:(intervalNotes a b)
```

- What's the largest departure from a whole note?
- How many new (normalised) notes are introduced?
- How far down the tone progression is the top note from the root?

Quality Measures

- How many distinct tones?
- are the generated tones Octave doublings of existing tones?
- [illegible]

```
  [x `elem` y | x <- map (normaliseNote . toNamedNote) [a, b],
let y = map (normaliseNote . toNamedNote) (intervalNotes a b)]
```

- How many new (normalised) notes are introduced?
- How far down the tone progression is the top note from the root?

Quality Measures

- How many distinct tones?
- are the generated tones Octave doublings of existing tones?
- What's the largest departure from a whole note?
- How many new (normalised) notes are

```
maximum . (0.0:) . map dissonance . map toNamedNote $ intervalNotes a b
```

- How far down the tone progression is the top note from the root?

Quality Measures

- How many distinct tones?
- are the generated tones Octave doublings of existing tones?

```
                  length $ filter not [x `elem` y |
      x <- map (normaliseNote . toNamedNote) (intervalNotes a b),
           let y = map (normaliseNote . toNamedNote) [a, b]]
```

- How many new (normalised) notes are introduced?
- How far down the tone progression is the top note from the root?

Quality Measures

- How many distinct tones?
- are the generated tones Octave doublings of existing tones?
- What's the largest departure from a whole note?
- [illegible] introduced?

```
max (derivationDepth a) (derivationDepth b)
```

- How far down the tone progression is the top note from the root?

Results

From C	D♭	D	E♭	E	F	G♭	G	A♭	A	B♭	B	C^1
Distinct Tones	3	3	3	3	2	4	2	3	3	3	3	1
Double Root/Top?	N/Y	Y/N	N/N	Y/N	N/Y	N/N	Y/N	N/Y	N/N	N/Y	Y/N	Y/Y
Max Dissonance	9e-3	3e-2	0	0	0	5e-2	0	0	0	6e-2	3e-2	0
Normalised New	1	1	2	1	0	2	0	1	2	1	1	0
Derivation Depth	4	4	2	2	2	6	2	3	2	4	4	1

Analyzing Spread

```
map (map toNamedNote . \(a,b,_) -> (intervalNotes a b)) $ allOf IIm
```

```
C-Db: Octave (-4) Db, Octave (-1) (Flat B (-9e-3))
Db-D: Octave (-5) (Flat B (-1e-1)), Sharp C 8e-3]
D-Eb: Octave (-4) Eb, Flat Db (-2e-2)
Eb-E: Octave (-5) Ab, Sharp D 2e-2
E-F: Octave (-4) F, Flat Eb (-3e-2)
F-Gb: Octave (-4) (Flat Eb (-5e-1)), Sharp E 8e-3
Gb-G: Octave (-4) G, Flat F (-1.6e-2)
G-Ab: Octave (-4) Ab, Flat Fs (-4e-3)
Ab-A: Octave (-4) Db, Sharp G 2.e-2
A-Bb: Octave (-4) Bb, Flat Ab (-3e-2)
Bb-B: Octave (-4) (Flat Ab (-4e-1)), Sharp A 8e-3
B-C: Octave (-3) C, Flat Bb (-2e-2)]]
```

Analyzing Spread

```
toneCounts = map (sum . map toneCount . allOf) intervals

toneCoincidences = map
  ((\(a, b) -> (length $ filter id a, length $ filter id b))
  . unzip . map ((\[a, b] -> (a, b)) . tonesCoincide)) $ map allOf
intervals

toneIntroductions = map (sum . map newTones) $ map allOf intervals

dissonanceCounts = map
  (sum . map (\(a, b, _) -> length . filter (>0) .
   map (dissonance . toNamedNote) $ intervalNotes a b) . allOf) intervals

numWithDissonance = map
  (length . filter (>0) . map (\(a, b, _) -> length . filter (>0) .
   map (dissonance . toNamedNote) $ intervalNotes a b) . allOf) intervals
```

Aggregate Results

From C	D ♭	D	E ♭	E	F	G ♭	G	A ♭	A	B ♭	B	C^1
Tone Counts	41	38	42	41	30	48	30	41	42	38	41	12
Tone Coincidences	0/7	6/0	0/0	7/0	0/9	0/0	9/0	0/7	0/0	0/6	7/0	0/0
Tone Introductions	17	18	24	17	6	24	6	17	24	18	17	0
Dissonance Counts	15	10	10	10	6	24	6	10	10	10	15	0
Num With Dissonance	12	8	6	6	3	12	3	6	6	8	12	0

The Interval Ranking

```
data Interval = IIm | II | IIIm | III | IV | Tri | V | VIm
         | VI | VIIm | VII | VIII deriving (Ord, Eq, Show)


intervals = [IIm, II, IIIm, III, IV, Tri, V, VIm, VI,
VIIm,
             VII, VIII]


intervalOrder = [V, IV, III, VIm, IIIm, VI, II, VIIm, IIm,
                 VII, Tri]


data RootLocation = Top | Bottom | Indeterminate


intervalRoots = [(IIm, Top), (II, Top), (III, Bottom),
  (IV, Top), (Tri, Indeterminate), (V, Bottom), (VIm,
Top),
  (VI, Top), (VIIm, Bottom), (VII, Bottom), (VIII,
Bottom)]
```

Chords

- Collections of notes

- Want to analyze
  - Chord Root
  - Chord Quality

Chord Root

1. Find best interval
   - *lowest* interval with *highest* ranking

2. Take root of best interval

3. There are exceptions
   - No IV or V, *and* a Tri
   - Repeated IV (and a VIIm)
   - Repeated III (and a VIm)

Finding Best Interval

```
notesToInterval first second =
    ordInterval (noteOrd (toNamedNote second) -
                 noteOrd (toNamedNote first))


normaliseChord notes = normaliseChord'
  (sortBy (\a b -> compare (pitch a) (pitch b)) notes) []
    where
      normaliseChord' [] _ = []
      normaliseChord' (h:t) norms =
        if (normNote `elem` norms)
        then normaliseChord' t norms
        else h:(normaliseChord' t ((normNote):norms))
          where
            normNote = normaliseNote . toNamedNote $ h
```

Finding Best Interval (cont.)

```
labelledChordIntervals notes =
  nubBy (\(a,_,_) (b,_,_) -> a == b) $
  sortBy (\(a,_,_) (b,_,_) -> compare a b)
  [(notesToInterval a b, a, b) |
    a <- notes', b <- notes' \\ [a]]
      where notes' = normaliseChord notes
```

```
bestLabelledInterval intervals =
  intervals !! (fromJust $ elemIndex interval intervals')
    where
      intervals' = map (\(a,_,_) -> a) intervals
      interval = bestInterval intervals'
```

Taking Root

```
chordRoot notes = if noRoot then Nothing else
             case lookup interval intervalRoots of
               Just Top -> Just top
               Just Bottom -> Just bottom
               _ -> Nothing
  where
    (interval, bottom, top) =
      bestLabelledInterval $ labelledChordIntervals notes
    intervals = chordIntervals notes
    noStrongRoot =
      length (intervals \\ [IIIm, VI, II, VII, IIm, VIIm])
      == 0
    diminishedTooUncertain =
      noStrongRoot && (Tri `elem` intervals)
    onlyFourths = length (intervals \\ [IV, VIIm]) == 0
    onlyThirds = length (intervals \\ [III, VIm]) == 0
    noRoot = onlyThirds || onlyFourths ||
diminishedTooUncertain
```

Chord Quality

Chords are sorted into 10 buckets

- Group A (No Tritone) vs. Group B (Tritone)
- AI / BII (no II, VII or IIm, VIIm)
- AIII / BIV (definite root)
- AV / BVI (no definite root)
- 1 (root is lowest note) vs. 2 (root is not lowest note)

Chord Quality

```
data ChordGroup = AI1 | AI2 | AIII1 | AIII2 | AV | BII1 | BII2 | BIV1 | BIV2 | BVI
deriving (Eq, Show)

chordGroup notes = if Tri `elem` intervals then chordB else chordA
  where
    intervals = chordIntervals notes
    chordA =  if (intervals \\ [II, IIm, VII, VIIm]) == intervals
              then  if chordRoot notes == Just (head notes)
                    then AI1
                    else  if chordRoot notes == Nothing
                          then AV
                          else AI2
              else  if chordRoot notes == Just (head notes)
                    then AIII1
                    else  if chordRoot notes == Nothing
                          then AV
                          else AIII2
    chordB =  if (intervals \\ [IIm, VII]) == intervals
              then  if chordRoot notes == Just (head notes)
                    then BII1
                    else  if chordRoot notes == Nothing
                          then BVI
                          else BII2
              else  if chordRoot notes == Just (head notes)
                    then BIV1
                    else  if chordRoot notes == Nothing
                          then BVI
                          else BIV2
```

The Minor Triad

- Two simplest chords are:
  - the major triad (I, III, V)
  - the minor triad (I, IIIm, IV)

- Major triad is easy to explain: generated by overtones 4, 5 and 6 of a root note.

- Minor triad is a major headache

The Minor Triad

- Mirror image of major triad?
  - (III, IIIm) ->(IIIm, III)
  - Needs justification as to why we can do this
  - Symmetry?
    - not a driving force in music
    - e.g. "major tonality" of major triad on I, IV, V is **not** mirrored by a "minor tonality"
  - Common overtone?
    - 6th over of I == 5th over of IIIm == 4th over of V
    - But overtones should be significant for major triad too
  - "Undertone series?"
    - Not a thing

The Minor Triad

Hindemith's approach equally poor

```
allIntervalNotes chord =
  nub . sortBy (\a b -> compare (pitch a) (pitch b)) .
  concat . concat $
  [[intervalNotes a b | b <- r] |
     value <- filter ((>1) . length) $ tails chord,
     let (a:r) = value]
```

- Major triad gives combination tones at:
  - [Octave (-2) C,Octave (-1) C,Octave (-1) G,C]
- Minor triad:
  - [Octave (-3) Ab,Octave (-2) Eb,Octave (-1) C, Octave (-1) Ab,Octave (-1) (Sharp Bb 2.5e-2)]

So what's going on?

http://www.audiotool.com/track/slide-8JjesTqb/

Stuff to think about

- Music
    - Better derivation of interval strength
    - Using combination tones to directly analyse chords

- Haskell
    - Secondary orderings
    - Cleaner extra-value threading

Questions?