A Functional Approach To Automatic Melody Harmonisation

Hendrik Vincent Koops
Faculty of Humanities,
Utrecht University
koops@phil.uu.nl

José Pedro Magalhães
Department of Computer Science,
University of Oxford
jpm@cs.ox.ac.uk

W. Bas de Haas
Department of Information and
Computing Sciences, Utrecht University
W.B.deHaas@uu.nl

Abstract

Melody harmonisation is a centuries-old problem of long tradition, and a core aspect of composition in Western tonal music. In this work we describe FHARM, an automated system for melody harmonisation based on a functional model of harmony. Our system first generates multiple harmonically well-formed chord sequences for a given melody. From the generated sequences, the best one is chosen, by picking the one with the smallest deviation from the harmony model. Unlike all existing systems, FHARM guarantees that the generated chord sequences follow the basic rules of tonal harmony. We carry out two experiments to evaluate the quality of our harmonisations. In one experiment, a panel of harmony experts is asked to give its professional opinion and rate the generated chord sequences for selected melodies. In another experiment, we generate a chord sequence for a selected melody, and compare the result to the original harmonisation given by a harmony scholar. Our experiments confirm that FHARM generates realistic chords for each melody note. However, we also conclude that harmonising a melody with individually well-formed chord sequences from a harmony model does not guarantee a well-sounding coherence between the chords and the melody. We reflect on the experience gained with our experiment, and propose future improvements to refine the quality of the harmonisation.

Categories and Subject Descriptors D.1.1 [*Programming Techniques*]: Functional Programming; H.5.5 [*Information Interfaces and Presentation*]: Sound and Music Computing

General Terms Languages, Human Factors

Keywords Harmony, Automatic harmonisation, Haskell, Haskore, HarmTrace, FHarm

1. Introduction

Ever since the Middle Ages, when singers in monasteries began to experiment with the addition of another voice to the chant, tentative efforts were made to discover laws that could accurately describe the way multiple voices should sound together. The sounding of multiple notes at the same time is called *harmony*, and the aspiring human mind brought forward theories to capture this phenomenon. With time, these rules changed, subject to fashion, politics, and even ecclesiastical law. With every generation, new rules are created, and some are rediscovered, rewritten, or forgotten.

Harmonisation is the process of finding a harmony for a given melody. For a single melody, multiple harmonisations can be created, but the number of possible harmonisations is constrained by the rules of tonal harmony. Harmonisation is a natural extension and application of *harmonic analysis*. Researching the steps involved in the process of harmonisation and automating these is a valuable contribution to the study of music, artificial intelligence, and Music Information Retrieval (MIR).

The research presented in this paper builds upon HARMTRACE, that formalises and implements the ideas of Rohrmeier (2007, 2011). HARMTRACE automatically derives the harmonic function of a chord in its tonal context given a sequence of symbolic chord labels (De Haas et al. 2013; Magalhães and De Haas 2011). Among other applications, these functional annotations can be used to improve the estimation of harmonic similarity of two chord sequences (De Haas et al. 2011), and to improve chord recognition from audio (De Haas et al. 2012).

This paper introduces FHARM, a system that uses HARMTRACE for complementing a melody with a chord sequence. After generating multiple possible chords for the given melody notes, we create a multitude of progressions. These progressions are subsequently fed into HARMTRACE to derive their functional harmony structures. From these progressions, we pick the one that deviates the least from the harmony model of HARMTRACE. This sequence is then optimised by an algorithm that determines the least varying chord inversions to diminish jumps in the harmony part. Finally, the harmony is combined with the input melody to create an output file.

We have chosen to implement FHARM in the Haskell programming language (Peyton Jones 2003). Haskell is a statically typed, lazy, purely functional programming language which allows for a very elegant and concise programming style. Instead of describing a computation in terms of statements that change a program's state, a functional language treats computation as the evaluation of mathematical functions, avoiding explicit state and mutable data. We believe Haskell is a good choice to implement rule-based functional harmonic analysis, in particular because of its support for algebraic datatypes, which naturally mimic grammar-like structures. The collection of datatypes in HARMTRACE can be viewed as a Context Free Grammar (CFG). The language defined by this CFG consists only of the combined values that match the structure of the datatype. This grammar defines a language of well-formed chord sequences, where the chord sequences are the values, and the types represent the relations between the structural elements.

The contribution of this paper is threefold. First, we formalise the problem of automatic harmonisation, and show how this problem can be approached in a functional and modular way. Second, we show how a formal model of Western tonal harmony can be used to help solve this problem. Last, we qualitatively evaluate the

FARM '13, September 28, 2013, Boston, MA, USA.

ACM 978-1-4503-2386-4/13/09. . . $15.00.
http://dx.doi.org/10.1145/2505341.2505343

FHARM system by reporting in-depth analyses of the automatic harmonisations by three independent experts. In FHARM, our aim is to produce an elegant melody harmonisation system that achieves good results in a simple, beautiful, and functional way. The entire code can be found at `http://dreixel.net/research/code/FHarm.tar.gz`.

The remainder of this paper is structured as follows. We start by covering some basic music theory needed to place the problem of harmonisation into context in Section 2, and we elaborate on harmonisation in particular in Section 3. Next, in Section 4 we explain the four main modules of FHARM: generating (Section 4.1), selecting (Section 4.2), parsing (Section 4.3), and post-processing (Section 4.4). In Section 5 we evaluate the automatic harmonisations of FHARM qualitatively. We conclude the paper discussing related work in Section 6 and future improvements of the system in Section 7.

2. About music theory

This section provides a brief introduction to music theory, needed to understand FHARM. For a thorough understanding of music theory, we refer the reader to a standard music theory textbook (such as Kostka et al. 2000). To ease the understanding of musical concepts for programmers, we accompany our description with illustrative code snippets of how to encode each musical concept in Haskell.

Within Western tonal music, sounds with a fixed frequency are commonly represented by a *note*. Notes have a name, which is usually denoted by one of the elements in the list $[C, C\sharp \approx D\flat, D, D\sharp \approx E\flat, E, F, F\sharp \approx G\flat, G, G\sharp \approx A\flat, A, A\sharp \approx B\flat, B]$. We can encode this in Haskell as follows:

```
data Note α          = Note Accidental α
data Accidental      = ♮ | ♯ | ♭
data DiatonicNatural = C | D | E | F | G | A | B
type NoteNatural     = Note DiatonicNatural
```

The ♯ denotes raising a note by a semitone, and the ♭ denotes lowering a note by a semitone. A ♮ means a note that is not altered. The semitone is the smallest interval used in Western music. In general, raising a note with a ♯ is *enharmonic equivalent* to that note one DiatonicNatural higher lowered by a ♭.[1] For example, *Note D* ♯ and *Note E* ♭ sound the same, but are "spelled" differently. We abbreviate the syntax of notes by writing $C\sharp$ instead of *Note* ♯ *C*, for example, and *C* instead of *Note* ♮ *C* (naturals are the default accidental). We denote enharmonic equivalence with $D\sharp \approx E\flat$, for example. We make Note a parametrised datatype because we will later have notes labelled with datatypes other than DiatonicNatural.

In musical set theory, *pitch classes* are defined by two equivalence relations. Pitches belong to the same pitch class if they have some relation of compositional or analytical interest, such as the octave relation (Roeder 2013). The second relation is the enharmonic equivalence relation, which means that all the pitches played on the same key of a regular piano keyboard are in the same set. From these two equivalence relations there are just 12 pitch classes, corresponding to the notes of the chromatic scale, often numbered from 0 to 11. The choice of which pitch class to call 0 is a matter of convention; we call *C* 0 (in which case $C\sharp \approx D\flat$ is 1, *D* is 2, etc.).

The distance between two notes is called an *interval*. An interval can be melodic or harmonic, in which the notes that make up that interval sound either consecutively or together, respectively. In Western tonal music, an interval is commonly classified as a combination between its quality (major, minor, or perfect) and number (unison, second, third, etc.). Common names for the intervals are shown in Table 1; enharmonic equivalent names have been omitted. A representation of intervals on a staff is shown in Figure 1.

Semitone distance	Name
0	unison
1	minor second (semitone)
2	major second
3	minor third
4	major third
5	perfect fourth
6	diatonic tritone
7	perfect fifth
8	minor sixth
9	major sixth
10	minor seventh
11	major seventh
12	octave

Table 1. A list of intervals and their names

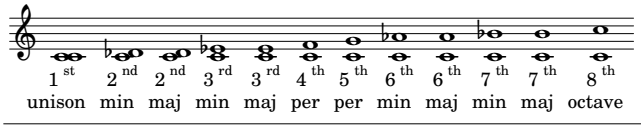

Figure 1. Intervals and their difference in staff positions.

An interval is *inverted* by moving one of the two notes an octave (or octaves) up or down so that they retain their pitch class. For example, one inversion of the interval consisting of a *C* with an *E* above it is an *E* with a *C* above it. To invert this interval, the *C* may be moved up or the *E* may be lowered. An interval together with its inversion yields an octave. This is because within an octave, a second inverts to a seventh, a third to a sixth, a fourth to a fifth, and so forth.

A sequence of notes in a specific ascending or descending order is called a *scale*. Scales are divided into categories based on the intervals they contain, being major, minor, or other. The first note of the scale is called the *tonic*, and the individual intervals between the notes of the scale and the tonic are called *scale degrees*. These scale degrees can be identified by Roman numerals (I, II, III, IV, V, VI, and VII). This scale degree representation can be implemented in Haskell as a DiatonicDegree together with Note in the following way:

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

Commonly, the notes of a scale belong to a certain *key*, and are the notes that sound reasonably well together. The key is named after the tonic, which is usually the focal point of a piece. A key is defined by the number of accidentals on the scale; Figure 2 shows which keys are associated with which specific accidentals.

The combination of three or more notes creates a *chord*. The simplest chords are called *triads*, and are created by superposing one interval of a third and one of a fifth on a common root note. If this third is minor, the *quality* of the chord is called *minor*; in the same way, if the third is major, the chord is called *major*. If the third is minor and the fifth is lowered one semitone, we call the chord *diminished*. The three notes of the chord are called root, third, and fifth.

Chords can be labelled unambiguously (Harte et al. 2005) by giving the following parts:

1. The chord root, which is either an absolute note like a *C* or a scale degree;

[1] This does not hold for all note combinations (in particular for *B* and *C*, and *E* and *F*), but further detail is unnecessary here.

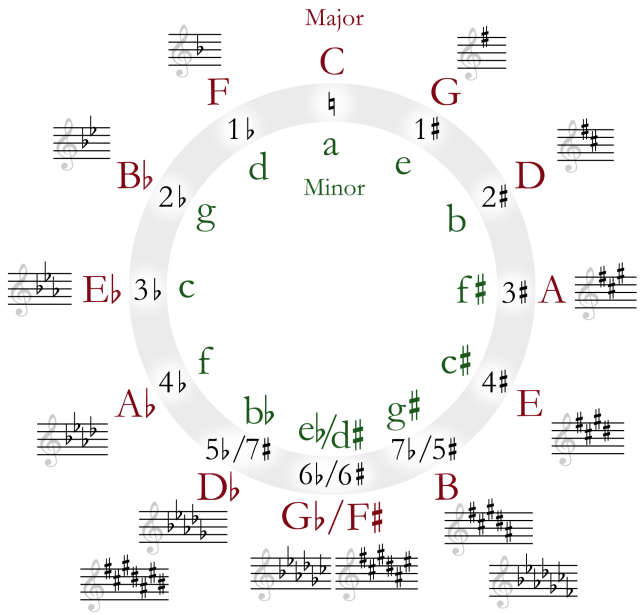

Figure 2. The circle of fifths.[2]

2. The quality of the chord, for example major or minor; and
3. Optional added or removed intervals, for example a seventh. For simplicity we will not use added intervals in our implementation.

We can encode chords in Haskell as follows:

```
data Chord α    = Chord {chordRoot      :: Note α
                        ,chordShorthand :: Shorthand}
data Shorthand = Maj | Min | Dim
```

This datatype represents a chord built from a *chordRoot*, which can be represented as a NoteNatural or a NoteDegree. This way we can represent chords built from an absolute root note, as well as chords built from scale degrees. Shorthand defines if the chord is major, minor, or diminished. Table 2 shows all the scale degrees and their corresponding chords in major and minor keys. We use *d*:*m* as a syntactic shorthand for *Chord d m*.

Functional harmony (Whittall 2013) is a theory of tonal harmony established by Riemann (1893), who devised the term. This theory describes the common harmony practice from the 18^{th} until the 20^{th} century. The functional harmony theory states that each chord within a key can be reduced to one of three harmonic functions—tonic, dominant, or subdominant. The tonic affirms the key, the subdominant builds tension, and the dominant builds maximum tension. These rules are expressed in the grammar of HARMTRACE, where it is defined that a subdominant is always followed by a dominant. This is explained in more detail in Section 4.3.1.

The *circle of fifths* is a geometrical representation of relationships among the twelve pitch classes of the chromatic scale (Figure 2). It shows how many flats or sharps a key has, and what its relative key is (the key with the same number of accidentals, but in the opposite mode). At the top of the circle is the C major key, which has neither sharps nor flats. In a clockwise direction, the notes are all a fifth apart (in an ascending perspective). Counter-clockwise, a circle of fourths is represented, since the inversion of a fifth is a fourth. This circle can be encoded in Haskell as follows:

```
circleOfFifths :: [NoteNatural]
circleOfFifths = [F♭,C♭,G♭,D♭,A♭,E♭,B♭
                 ,F  ,C  ,G  ,D  ,A  ,E  ,B
                 ,F♯,C♯,G♯,D♯,A♯,E♯,B♯]
```

3. Harmonisation

> *Harmony is the study of simultaneous sounds (chords) and of how they may be joined with respect to their architectonic, melodic, and rhythmic values and their significance, their weight relative to one another.*
>
> Arnold Schönberg - *Theory of harmony (Schönberg 1978)*

In music in general, the basic element is the note. In harmony, the basic unit is the *interval*. Harmonisation is not subject to universal laws, as all eras and even individual composers have developed their own harmony practices. This section describes the harmonisation rules that are generally applicable in Western tonal music, as presented by Piston and DeVoto (1991, considered to be a classic reference in the field).

Harmonising a melody is the process of complementing a given melody with chords. Any melody can be harmonised in at least one way. Hence, harmonisation can best be viewed as the creative process of selecting the best matching chord for a melody segment. This process consists roughly of three consecutive stages: melody analysis, chord selection, and sequence selection.

Before chords can be considered, an analysis of the melody is needed. Knowing the time signature and tonality of the melody is of vital importance for creating a functional harmony. The time signature is the conventional way in Western musical notation to specify how many beats are in each bar and which note value constitutes one beat. This can be implemented using a Rational type, represented as a ratio of two Integer values:

```
type TimeSig = Rational
```

For example, a time signature of $\frac{4}{4}$ is encoded as $4 \% 4$, where the numerator is the number of beats, and the denominator is the beat note value.

The tonality of the melody tells us which chords can be used with the individual notes, while the time signature tells us something about the occurrence of the chords, as the strong metrical positions are treated with more importance with regards to chord placement. Segmenting the melody in parts in concordance with the time signature tells us where and how many chords can be used. The structure of the melody is also a guide when choosing chords. With large melodic skips ending on weak bar structures, it is preferred to use the same harmony for both notes. Likewise, if a note is sustained, the chord is usually not changed.

After analysing the melody, chords should be selected in line with the notes. Because a single note in the major or minor scale occurs in three common triads of the scale, a melody of n notes can be harmonised in 3^n ways when using common key chords (i.e. chords that contain the melody note). Quite a few of these sequences will sound unnatural and break many common harmony theory rules, but, for our purposes, we will consider them harmonisations of that melody nonetheless.

The last phase is sequence selection. Constructing a sequence out of the 3^n possibilities involves selecting chords that follow a logical, naturally sounding line. Several heuristics can be used for

[2] Image taken from `http://en.wikipedia.org/wiki/File:Circle_of_fifths_deluxe_4.svg`.

Scale degree	*I*	*II*	*III*	*IV*	*V*	*VI*	*VII*
Major key	*I*:*Maj*	*II*:*Min*	*III*:*Min*	*IV*:*Maj*	*V*:*Maj*	*VI*:*Min*	*VII*:*Dim*
Minor key	*I*:*Min*	*II*:*Dim*	*III*:*Maj*	*IV*:*Min*	*V*:*Min*	*VI*:*Maj*	*VII*:*Dim*

Table 2. Scale degrees and chords arising in major and minor keys.

this purpose. For example, a harmonisation should end in a *cadence*: a harmonic progression of at least two chords that gives the listener a sense of conclusion of a phrase. Common cadences are the authentic and the half cadence. The authentic cadence is a progression of *V*–*I*, or *IV*–*V*–*I*. This is considered a strong cadence, as it affirms the tonic and gives the listener the feeling that the phrase is complete. The half cadence is any cadence ending on *V*, preceded by any chord. Because it sounds incomplete or "suspended", the half cadence is considered a weak cadence. A few more cadences exist, but, for simplicity, we only consider these two types.

Cadences can be built from right to left. For example, in the key of *C*, with a melody ending on the tonic, any of the chords *I*:*Maj*, *IV*:*Maj*, or *VI*:*Min* can be chosen to harmonise the last note, because *C* appears in all of those chords. If ending on an authentic or half cadence is the goal, only *I* and *V* are valid choices. In this case, the cadence should be an authentic one, because *I* is available, and *V* is not, eliminating *IV* and *VI* from the possible choices. Knowing that the sequence will end on *I*, a perfect cadence can be made by choosing *V* as the chord before the last. If the note before the *V* allows for a *IV* chord, the *IV* can be chosen as well, creating the long form of the authentic cadence.

A couple more examples of heuristics are: if notes in a bar share the same chord, the first chord of the bar can be sustained. If a melody begins on a strong bar part, a *I* is chosen as the first chord; if it begins on a weak bar part, a *V* is chosen as the starting chord of the sequence. These are just a few of the many of rules the can be applied while selecting harmonic sequences from all possibilities.

4. FHARM

This section introduces FHARM, an extension of HARMTRACE for harmonising melodies. It extends HARMTRACE by adding a knowledge-based system that selects well-matching chord sequences according to predefined rules. Using HARMTRACE, given a sequence of chord labels generated by FHARM, the harmonic function of a chord in its tonal context is automatically derived.

The handling of the input MIDI file is done with HASKORE (Hudak et al. 1996) and the `Sound.Midi` libraries (Thielemann 2004). HASKORE uses a simple algebraic approach to music description and composition. In HASKORE, musical objects consist of primitive notions such as notes and rests, and operations to transform musical objects to create more complex ones, such as concurrent and sequential compositions (chords and melodies, respectively). The `Sound.Midi` library is used for reading and writing MIDI files, so that the output of our algorithm can be heard.

FHARM is made up of four independent, composable steps:

1: Generating Analysing the melody and generating candidate chords for each melody note;

2: Selecting Filtering the generated chord candidates through a rule-based system to prevent unwanted and illogical sequences;

3: Parsing Determining the best harmonisation by feeding the generated sequences to HARMTRACE, favouring analyses with fewer errors;

4: Post-processing Processing the sequence to find the least varying chords inversions along a trend line, and combining these with the input melody into an output MIDI file.

We describe each of these steps in the following subsections.

4.1 Generating

FHARM takes a single track, monophonic MIDI file as its input. We use the `Sound.Midi` library to extract tonal and rhythmic information. We then use HASKORE to represent the MIDI file contents as datatypes encoding tonal information.

In this first phase, each note of the the melody is transformed into a value of type MelodyNote, which represents each note as a 4-tuple containing a Root, Octave, Duration and ChordList:

```
type MelodyNote = (Root, Octave, Duration, ChordList)
type Root       = NoteNatural
type Octave     = Int
type Duration   = Rational
type ChordList  = [Chord NoteDegree]
```

Root represents the current melody note as an absolute root note in terms of the NoteNatural definition that was given earlier (Section 2). Octave is defined as an integer number which denotes the octave the melody note is in. Duration represents the length of the note as ratios of two Int values, which encodes the relative duration of a note. ChordList is a list of candidate chords that can be used to harmonise this note. In this initial phase, ChordList is an empty list.

The entire melody represented as multiple MelodyNotes is encoded as a value of the Song datatype:

```
data Song = Song { key      :: Key
                 , timesig  :: TimeSig
                 , leap     :: Leap
                 , melody   :: [(Bar, [MelodyNote])] }
type Bar  = Rational
type Leap = Rational
```

In this encoding, only the properties of the input file necessary to derive a harmonisation are represented. Key denotes the key of the melody, and is passed as an argument to FHARM. It consists of a Root, and a Mode, which denotes whether the Key is major or minor:

```
data Key  = Key { keyRoot  :: Root
                , keyMode  :: Mode }
data Mode = Maj_Mode | Min_Mode
```

The time signature of the MIDI file can be derived by reading the appropriate metadata fields. Leap represents the ratio of chords per beat. A leap value of s tells us that every s^{th} beat should have a chord. This notion is explained further in Section 4.2.1. The melody notes of the song are defined in terms of the Bar and MelodyNote types. Bar represents the current bar, starting at (1,n) and ending with (n,n), where n is the number of bars of the melody. This notion is used to keep track of which notes occur in which bar. The second element of the pair is a list of MelodyNotes as defined before.

4.1.1 Melody to pitch class

To select three possible chords for each melody note, the absolute notes are first converted to absolute integer values. This representation uses no key information, but represents the note as absolute integer values of the chromatic scale, being the position in the list $[C, C_\sharp \approx D_\flat, D, D_\sharp \approx E_\flat, E, F, F_\sharp \approx G_\flat, G, G_\sharp \approx A_\flat, A, A_\sharp \approx B_\flat, B]$. We implement this as follows:

```
diaNatToSemi :: Root → Int
diaNatToSemi (Note m n) =
    [0,2,4,5,7,9,11] !! (fromJust $ elemIndex n [C ..])
  + modToSemi m
modToSemi :: Accidental → Int
modToSemi ♮ =  0
modToSemi ♯ =  1
modToSemi ♭ = −1
```

Given a Root representation of a note, *diaNatToSemi* returns its absolute integer value. Note that we're making use of a (trivial) instance of Enum for DiatonicNatural.

4.1.2 Chord candidate generation

To find candidate chords that contain a specific pitch class, we build chord structures representing the notes that are associated with a chord. All chords have a specific tonal content; for example, every major chord consists of a major third and a perfect fifth on top of a root note. We can thus represent chords as a binary list of occurring notes; as an example, here is the representation of the *C*:*Maj* chord:

$$\begin{array}{c} C, D_\flat, D, E_\flat, E, F, F_\sharp, G, A_\flat, A, B_\flat, B \\ [1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0] \end{array}$$

To build a major chord on the root note *D*, for example, we rotate the list two positions:

$$\begin{array}{c} C, D_\flat, D, E_\flat, E, F, F_\sharp, G, A_\flat, A, B_\flat, B \\ [0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0] \end{array}$$

By representing chords as a binary list we abstract from the root note, which means we can analyse chord structures without having to define all major, minor, and diminished chords of a key individually:

```
shortHandToChordStructure :: Shorthand → [Integer]
shortHandToChordStructure sh =
  case sh of
    Maj → [1,0,0,0,1,0,0,1,0,0,0,0]
    Min → [1,0,0,1,0,0,0,1,0,0,0,0]
    Dim → [1,0,0,1,0,0,1,0,0,0,0,0]
```

We can now easily find candidate chords for each melody note. Because a triad chord contains three notes of the scale (root, third, and fifth), there are three possible chords for each note: one that has the melody note as its root, one that has the melody note as its third, and one that has the melody note as its fifth.

We list all the appropriate chords for a given key and mode, and convert them to their corresponding binary representation. This gives us a list of binary chord representations, each one rooted on each scale note. To find the allowed chords for a note from this list, a 1 must be present at position *i* of the binary representation of that chord, where *i* is the chromatic integer representation of the note. At this stage, we expand the ChordList in the Song datatype with the three chords that are allowed for each note. For example, for the melody note *C* in the key *C*, the ChordList is expanded to $[I{:}Maj, IV{:}Maj, VI{:}Min]$, as each of those chords contains *C*.

4.2 Selecting

After having generated candidate chords for each melody note, we proceed to filter these sequences to avoid unusual chord progressions.

4.2.1 Eliminate unwanted sequences

To reduce the search space, we enforce the notion of a cadence: since a half cadence ends in *V*, and an authentic cadence ends in *I*, the list of possible chords of the last melody note is reduced to these cadence-only chords. Additionally, since it is uncommon to start a chord sequence on a chord other than *I* or *V*, a similar filtering method is used for the chords of the first melody note.

4.2.2 Removing leap chords

When harmonising a melody it is custom to not assign a chord to every note. Doing so would distract from the rhythm of the melody and make the musical piece sound mechanical. Usually, chords are placed at strong metrical positions, which occur on the beat. The third value of the Song datatype is *leap*, represented as a ratio. This number defines how many notes per bar in the melody will be harmonised with a chord. This *leap* value is passed as a parameter to FHARM, and tells us that for a leap value *l* there should be a chord at every l^{th} beat of a bar, and the other notes should have no candidate chords.

To calculate which notes will be harmonised with a chord, we simply iterate through the melody in each bar, and add up the duration of the notes until we have reached the *leap* value. If the note occurs on the l^{th} beat, its ChordList stays intact; otherwise it is replaced by an empty list. It might seem strange that we first generate chords for every melody note, and then remove some, while we could directly generate chords only for the notes we are interested in. However, this is more modular; our current mechanism for removing leap chords is rather simplistic, but a more advanced implementation could take into account the surrounding chords proposed for melody notes that do not get their own chord, for example.

4.2.3 Chord probability assignment

Even though several chords might fit a given melody note, they are not all equally well-suited. In FHARM, the notion of how well a chord fits a melody note is defined using the distance of that note to the root note of the chord in the circle of fifths. We encode this notion by associating probabilities with each chord. We use the circle of fifths because the tonal distances represented there correspond better to human perception than a simple linear distance.

For example, in the key of *C*, the *E* note has three possible chords for harmonisation: *C*:*Maj*, *E*:*Min*, and *A*:*Min*. The root notes of these chords are respectively *C*, *E*, and *A*. The distance in the circle of fifths between these root notes and the melody tone *E* are 4, 0, and 1. We calculate the likelihood for each chord by dividing these numbers by 10 and subtracting them individually from 1. The list of chords with their likelihood is obtained by converting the chords to a Chord DiatonicDegree notation and creating list of tuples containing both the chord and its likelihood. In our example, this list is $[(I{:}Maj, 0.6), (III{:}Min, 1.0), (VI{:}Min, 0.9)]$. This is a value of type ProbChordList:

```
type ProbChordList = [(Chord DiatonicDegree, Double)]
```

The values in ProbChordList are then normalised in the interval $[0..1]$ to obtain actual probabilities that are used in a random selection mechanism. We now explain in detail how these lists are calculated.

Converting melody notes To calculate the distance in the circle of fifths, we need to be able to compare the melody note to the chord root note. To do this, we need to first convert the Integer number representation of the melody note to a Note DiatonicNatural. This is easily done by using the Integer number representation as an index of the list of pitch classes.

Converting chord root notes The other note needed to calculate the distance in the circle of fifths is the root note of the chord. This is relatively easy, as the Chord NoteDegree (Roman numeral) representation of the chord can be transformed into a

Chord NoteNatural (note name) representation. The Roman numeral of the scale degree is changed into its Arabic numeral equivalent minus 1, which is used as the position to lookup in a list of notes for the key. For instance, the key of *A* consists of the notes $[A,B,C,D,E,F,G]$. The chord with the Roman numeral *III* has an Arabic equivalent of 3, and at position $3-1 \equiv 2$ of the list we find the note *C*.

Distance in the circle of fifths The function that calculates the distance in the circle of fifths simply finds both notes in the *circleOfFifths* list (of Section 2) and returns the distance between their positions. Now that the distance *d* between a melody note and the root note of a possible chord can be calculated, we assign a likelihood $1-d\,/\,10$ to that chord. After calculating this likelihood for each chord, the values are normalised to probabilities in the interval $[0\,..\,1]$.

For the note *E* in the key of *C*:*Maj*, for example, the following list is created: $[(I{:}Maj,0.6),(III{:}Min,1.0),(VI{:}Min,0.9)]$. To give each chord a probability between 0 and 1, each of the likelihood values is divided by the sum of all the values (2.5). This gives us the relative intervals of the probabilities of the chords, which we then sort by lowest probability first, to obtain the list $[(I{:}Maj,0.24),(VI{:}Min,0.36),(III{:}Min,0.4)]$.

4.2.4 Sequence generation

The calculated probabilities of the chords are used to create chord sequences. A random selection algorithm decides which chord is chosen for a melody note. We explain this continuing the example given previously. We generate a uniformly distributed random number between 0 and 1. If this number is $\leqslant 0.24$, chord *I*:*Maj* is chosen. If the number is $0.24 < x \leqslant 0.24+0.36 \equiv 0.6$, then chord *VI*:*Min* is chosen. Lastly, if the number is >0.6, chord *III*:*Min* is returned. We encode this in Haskell as follows:

```
sample :: [(Chord α, Double)] → IO Int
sample l = do p ← getStdRandom (randomR (0.0, 1.0))
              return (pick p l)

pick :: Double → [(Chord α, Double)] → Int
pick p = fromJust ∘ findIndex (>p) ∘ tail
       ∘ scanl (λa b → a + snd b) 0
```

Repeating this process for each melody note, we generate a sequence of chords that harmonises the melody. We then repeat this process a fixed number of times over the entire melody, to generate several lists of chords, each representing harmonisations over the input melody. These harmonisations are then passed on to the next stage.

4.3 Parsing

To ensure harmonic coherence in the generated chord sequences, we parse each sequence using HARMTRACE. In this section we introduce the harmony model of harmtrace, and how chord sequences are parsed to fit this model. This is mostly restating previous work (De Haas et al. 2011, 2012, 2013; Magalhães and De Haas 2011), adapted to the purposes of FHARM.

4.3.1 Harmony model

HARMTRACE contains a harmony model that implements and extends the ideas of Rohrmeier (2007, 2011), who modelled the core rules of Western tonal harmony as a (large) context-free grammar (CFG). HARMTRACE models tonal harmony as a Haskell datatype. Since this model affects the quality of our generated harmonisations, we explain it briefly in this section. For clarity, we chose a syntax that closely resembles a CFG. A CFG defines a language: it accepts only combinations of words that are valid sequences of the language. A collection of Haskell datatypes can be viewed as a CFG: the type-checker accepts a combination of values if their structure matches the structure prescribed by the datatype, and rejects this combination if it does not. Within HARMTRACE, the chords are the values, and the datatypes represent the relations between the structural elements in tonal harmony. We show some example analyses produced by HARMTRACE in Section 5.

We start by introducing a variable (denoted with Fraktur script) for the mode of the key of the piece, which can be major or minor. The mode variable is used to parametrise all the specifications of our harmony model; some specifications hold for both modes ($\mathfrak{M}$), while other specifications hold only for the major (Maj) or minor mode (Min). The mode is displayed as a subscript, which we leave out when it is clear from the context. In Haskell, these variables are implemented as indices in a generalised algebraic datatype (Schrijvers et al. 2009); a specification that constrains a variable corresponds to a constructor that introduces an equality constraint on the index, while a specification that does not constrain a variable corresponds to a constructor with no constraints on the index.

Spec. 1–3 define that a valid chord sequence, $Piece_{\mathfrak{M}}$, consists of at least one and possibly more functional categories. A functional category classifies chords as being part of a tonic ($Ton_{\mathfrak{M}}$), dominant ($Dom_{\mathfrak{M}}$), or subdominant ($Sub_{\mathfrak{M}}$) structure, where a subdominant must always precede a dominant. These functions constitute the top-level categories of the harmonic analysis and model the global development of tonal tension: a subdominant builds up tonal tension, the dominant exhibits maximum tension, and the tonic releases tension. The order of the dominants and tonics is not constrained by the model, and they are not grouped into larger phrases:

$$1\ \ Piece_{\mathfrak{M}} \rightarrow [Func_{\mathfrak{M}}]$$
$$2\ \ Func_{\mathfrak{M}} \rightarrow Ton_{\mathfrak{M}} \mid Dom_{\mathfrak{M}} \qquad \mathfrak{M} \in \{\mathrm{Maj},\mathrm{Min}\}$$
$$3\ \ Dom_{\mathfrak{M}} \rightarrow Sub_{\mathfrak{M}}\ Dom_{\mathfrak{M}}$$

Spec. 4–8 translate dominants, tonics, and sub-dominants into scale degrees (denoted with Roman numerals). A scale degree is a datatype that is parametrised by a mode, a chord class, and the interval between the chord root and the key. The chord class is used to constrain the application of certain specifications, e.g. Spec. 13 and 14, and can represent the class of major (no superscript), minor (*m*), dominant seventh (7), and diminished seventh chords (0). A tonic translates into a first degree in both major and minor mode, albeit with a minor triad in the latter case, or it allows for initiation of a plagal cadence. A dominant type is converted into the fifth scale degree with a dominant or diminished class, respectively. Similarly, a sub-dominant is converted into the fourth or second degree:

$$4\ \ Ton_{\mathrm{Maj}} \rightarrow I_{\mathrm{Maj}} \mid I_{\mathrm{Maj}}\ IV_{\mathrm{Maj}}\ I_{\mathrm{Maj}}$$
$$5\ \ Ton_{\mathrm{Min}} \rightarrow I^{m}_{\mathrm{Min}} \mid I^{m}_{\mathrm{Min}}\ IV^{m}_{\mathrm{Min}}\ I^{m}_{\mathrm{Min}}$$
$$6\ \ Dom_{\mathfrak{M}} \rightarrow V^{7}_{\mathfrak{M}} \mid V_{\mathfrak{M}}$$
$$7\ \ Sub_{\mathrm{Maj}} \rightarrow IV^{m}_{\mathrm{Maj}} \mid II^{m}_{\mathrm{Maj}} \mid \ldots$$
$$8\ \ Sub_{\mathrm{Min}} \rightarrow IV_{\mathrm{Min}} \mid II^{m}_{\mathrm{Min}} \mid \ldots$$

Finally, scale degrees are translated into the actual surface chords that are used as input for the model. The chord notation used is that of Harte et al. (2005). The conversions are trivial and illustrated by a small number of specifications below. The model uses a key-relative representation, and in Spec. 9–12 we used chords in the key of C. Hence, a I_{Maj} translates to the set of C chords with a major triad, optionally augmented with additional chord notes that do not make the chord minor or dominant. Similarly, V^{7}_{Maj} translates to all G chords with a major triad and a minor seventh, etc.:

$$9\ \ I_{\mathrm{Maj}} \rightarrow \texttt{"C:maj"} \mid \texttt{"C:maj6"} \mid \texttt{"C:maj7"} \mid \ldots$$

10 $I^m_{\text{Min}} \rightarrow$ `"C:min"` | `"C:min7"` | `"C:min9"` | ...

11 $V^7_{\mathfrak{M}} \rightarrow$ `"G:7"` | `"G:7(b9,13)"` | `"G:(#11)"` | ...

12 $VII^0_{\mathfrak{M}} \rightarrow$ `"B:dim(bb7)"`

Spec. 13 accounts for the classical preparation of a scale degree by its secondary dominant, stating that every scale degree, independently of its mode, chord class, and root interval, can be preceded by a chord of the dominant class, one fifth up. The function $V/\mathfrak{X}$ (implemented as a type family) transposes an arbitrary scale degree $\mathfrak{X}$ a fifth up. Similarly, every scale degree of the dominant class can be prepared with the minor chord one fifth above (Spec. 14):

13 $\mathfrak{X}^{\mathfrak{C}}_{\mathfrak{M}} \rightarrow V/\mathfrak{X}^7_{\mathfrak{M}}\ \mathfrak{X}^{\mathfrak{C}}_{\mathfrak{M}} \qquad \mathfrak{C} \in \{\varnothing, m, 7, 0\}$

14 $\mathfrak{X}^7_{\mathfrak{M}} \rightarrow V/\mathfrak{X}^m_{\mathfrak{M}}\ \mathfrak{X}^7_{\mathfrak{M}} \qquad \mathfrak{X} \in \{I, II\flat, II, \ldots, VII\}$

The harmony model in HARMTRACE further allows various scale degree transformations. Every dominant chord can be transformed into its tritone substitution with Spec. 15. This specification uses another transposition function $V\flat\,/\,\mathfrak{X}$ which transposes a scale degree $\mathfrak{X}$ a diminished fifth (a tritone) up. Likewise, diminished seventh chords are treated as regular dominant seventh chords without a root and with a $\flat 9$ (Spec. 16). For instance, an $A\flat^0$, consisting of A$\flat$, B, D, and F, is viewed as a $G^{7\flat 9}$, which consists of G, B, D, F, and $A\flat^0$. An exceptional characteristic of diminished seventh chords—consisting only of notes separated by minor third intervals—is that they are completely symmetrical. Hence, a diminished seventh chord has four enharmonic equivalent chords that can be reached by transposing the chord a minor third up with the transposition function $III\flat/\mathfrak{X}$ (Spec. 17):

15 $\mathfrak{X}^7_{\mathfrak{M}} \rightarrow V\flat/\mathfrak{X}^7_{\mathfrak{M}}$

16 $\mathfrak{X}^7_{\mathfrak{M}} \rightarrow II\flat/\mathfrak{X}^0_{\mathfrak{M}}$

17 $\mathfrak{X}^0_{\mathfrak{M}} \rightarrow III\flat/\mathfrak{X}^0_{\mathfrak{M}}$

We have presented a condensed view on the core specifications of the model. Due to space constraints we had to omit some specifications for diatonic chains of fifths, borrowings from the parallel mode, and the Neapolitan chord. For the full specification of the model, we refer the reader to De Haas et al. (2013).

4.3.2 From textual chord labels to structured harmony

Having a formal specification of the rules of tonal harmony as a Haskell datatype, the next step is to use a parser to transform textual chord labels into values of this datatype. The HARMTRACE parser makes use of *datatype-generic programming* techniques to avoid writing most of the repetitive portions of the code. Moreover, not only the parser can be derived automatically, but also a pretty-printer for displaying the harmony analysis in tree form, and functions for comparing these analyses. For technical details of the implementation and use of generic programming techniques, we refer the reader to Magalhães and De Haas (2011).

Since HARMTRACE cannot account for all possible harmony rules ever used, and also because it is designed to be robust against noisy or incorrect chord sequences, it uses an error-correcting parser (Swierstra 2009) that automatically adds and removes chords from the sequence in order to create a well-typed value. We use the number of errors during parsing as a way to sort the progressions. For most progressions, parsing results in little or no errors. Progressions with many surprising or unexpected chords, however, give rise to multiple error-correction steps. This strategy serves to impose a form of "quality control" on our generated chord sequences. When multiple trees have the same minimum number of errors, we pick the one with the simplest harmony analysis (the one with fewest nodes).

4.4 Post-processing

The final step before creating a MIDI file with the melody and the chord progression is to adapt the generated harmony by picking chord inversions that minimise the distance between chord pitches. For this we again use HASKORE (Hudak et al. 1996). This step helps keeping the overall pitch of the chords as close together as possible, resulting in a smoother harmonisation. The algorithm computes the centre of each chord, which is the average of all its pitches. Using these centres, it then tries to keep the centre as close as possible to an overall trend. The adapted chord progression is then merged with the original melody, and output to a MIDI file.

5. Experiments and results

To measure the quality of our harmonisations, we have carried out two experiments. In experiment one, we confronted three experts in harmony theory with harmonisations of melodies created by FHARM. The experts were requested to give their professional opinion on the technical and creative aspect of the horizontal and vertical dimensions of the harmonisation. In the second experiment, for a given melody, we compare a harmonisation created by a harmony expert with the outcome given by FHARM. Both experiments use melodies given in exercises of Piston and DeVoto (1991). The inputs used to FHARM in this section and sample output harmonisations are available at `http://dreixel.net/research/code/faamh_examples.zip`.

5.1 Experiment one

In this experiment, three experts were confronted with melodies harmonised by FHARM. The harmonised melodies were presented to them in score form as well as their corresponding audible MIDI file. The melodies chosen for this survey are exercises b, c, and e from Chapter 7 (*Harmonisation of a given part*) of Piston and DeVoto (1991). We will refer to these melodies as melody A, melody B, and melody C, respectively. These melodies were chosen because of their difference in key, time signature, and overall structure.

The panel of experts consisted of:

Expert I an undergraduate student in musicology at Utrecht University;

Expert II a composer in pop-oriented music, who studied composition and sound design;

Expert III a music lecturer at the Utrecht School of the Arts, faculty of art, media, and technology.

We have purposely included experts from different professional musical areas to see if they would derive different conclusions from listening to the same pieces. Although we can observe some difference in the analyses, there is a general agreement in the broad sense of the appreciation and technical aspects of the pieces.

5.1.1 Method

For this experiment, FHARM generated one harmonisation for each melody (out of 1000 candidate harmonisation per melody), as described in Section 4.

Three questions pertaining to the choice of the individual chords per melody tone and their overall development over time were given. The first question inquires about the technical aspect, or correctness of the total progression, expressed in a number from 1 to 5 (1 being very correct, and 5 being very incorrect). In addition to this question, experts are asked to explain this rating in their own words.

The second question inquires about the choice of the individual chords in the harmonisation, asking experts to rate every chord on a

scale from 1 to 5 (1 being very correct, and 5 being very incorrect). This rating represents whether that chord fits the melody segment it belongs to. In addition to this rating, experts are asked what chords they would replace, and in what way. The third question is in open form, inquiring whether the expert has anything more to say about the harmonisations in general. The question form used in the experiment is transcribed in Appendix A.

5.1.2 Melody A

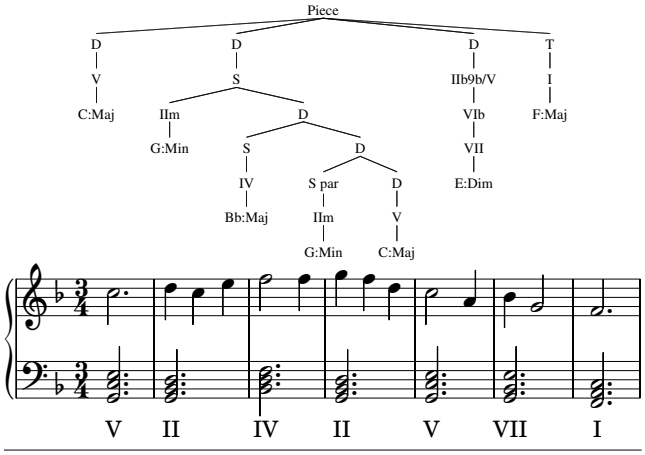

Figure 3. Melody A in F major with harmonisation (as generated by FHARM) and its harmonic analysis (as generated by HARMTRACE).

Melody A and its generated harmonisation can be seen in Figure 3, and the experts' ratings in Figure 4. As can be seen from the results, the progression of the second to third chord (*II - IV*) is found to be troubling. In particular, the movement between the two chords has too many voices moving in the same direction. When moving from *II* to *IV* in *F* major, the algorithm to determine the shortest pitch distance (in terms of chord inversions) did a good job, but in this case it leads to a bad choice. In classical harmonisation, parallel movement (placing two chords in succession such that every note moves with the same interval) is considered bad practice. In this case every note in *II* is a third lower than the *IV* chord after it. The inversion algorithm should make a different decision here, or the choice of another chord should be made, to avoid parallel movement of chords.

This harmonisation is found to be the best out of the three melodies; it was rated by expert I as 2, II as 1, and III as 3, and overall well sounding. Although not in accordance with the classical rules, the progression is regarded as a functional one: one that *works*.

5.1.3 Melody B

Melody B and its generated harmonisation can be seen in Figure 5, and the experts' ratings in Figure 6. This melody has the least favoured harmonisation by the experts, and was criticised with regard to the overall structure and progression.

According to the experts, this harmonisation seems to lack a connection to the melody in certain places. Although they agreed that individual chords fit the notes well, substitution of certain chords would make the progression stronger. Interestingly, the functional analysis of the progression as shown in the tree in Figure 5 shows little inner structure. An explanation for the lack of coherence in the progression could be attributed to the lack of functional components that are built out of multiple chords, or chord sequences. This way, the chords of the sequence have little functional relationship with each other, except for their shared key.

	V	II	IV	II	V	VII	I	Overall
I	1	2	3	1	1	1	1	2
II	1	1	2	1	1	1	1	1
III	1	2	5	2	2	2	1	3
Average	1	1.7	3.3	1.3	1.3	1.3	1	2

Figure 4. Melody A: ratings of the experts of the individual chords and overall progression on a scale from 1 to 5 (second and first questions of the survey, respectively; lower scores are better).

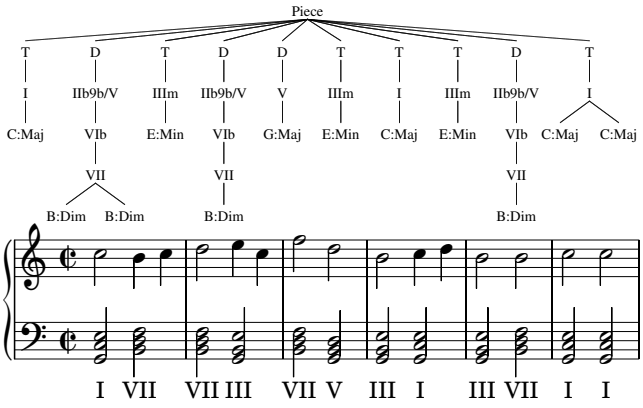

Figure 5. Melody B in C major with harmonisation (as generated by FHARM) and its harmonic analysis (as generated by HARMTRACE).

The experts found that a better choice could be made in the second bar, where a *III* chord is found as a harmonisation of the note *E*. The choice for a *III* chord can be attributed to the fact that the chord generating algorithm takes only the current note into account, and ignores the rest of the bar (the note *C*). The *III* chord does not *resolve* the *C*, which in this case means using a chord that has a *C* in it. The notion of resolution is important in classical harmony theory, as it defines the passing of dissonant sounds into consonant sounds during the course of changing harmony. In this case, the note *C* creates a *dissonance* with the *III* chord.[3] This is because the *III* chord contains a *B*, which together with *C* is a minor second. A relatively easy way to improve FHARM would be to use all notes in a bar for choosing a chord instead of just the current one. This could create harmonisations that are less disjunct from the melody.

Another remark is that the centre of gravity in the harmony is not in the right place. This can be attributed to the fact that the chord generation system does not take into account the overall arc of the melody. As the algorithm focuses on the current note of the melody, it is unaware of notions like the direction (up or down) in which the melody is going, or more complex structures like a recurrent piece of the melody (theme).

The HASKORE algorithm for calculating the least varying chord inversions makes good choices as far as the shortest overall distance between chords is considered, but is too strict in the sense of classical harmonisation. When all notes between chords move in the same direction with the same interval, like they do in bars 2–3, it is better to pick a different inversion, even if that one is further away.

Different inversions of the same chord have different functions. The second inversion of a chord, in which the root note is moved

[3] A dissonance is any interval other than unison, octave, perfect fourth or fifth, and major or minor third and sixth.

	I	VII	VII	III	VII	V	III	I	III	VII	I	I	Overall
I	1	3	1	4	1	1	1	4	2	2	2	2	4
II	1	3	5	3	3	1	2	3	2	1	1	1	3
III	3	3	3	5	5	5	3	2	3	2	5	5	4
Average	1.7	3	3	4	3	2.3	2	3	2.3	2	2.7	2.7	3.7

Figure 6. Melody B: ratings of the experts of the individual chords and overall progression on a scale from 1 to 5 (second and first questions of the survey, respectively; lower scores are better).

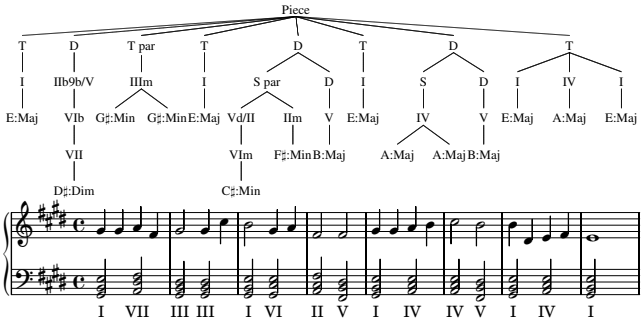

Figure 7. Melody C in E major with harmonisation (as generated by FHARM) and its harmonic analysis (as generated by HARMTRACE).

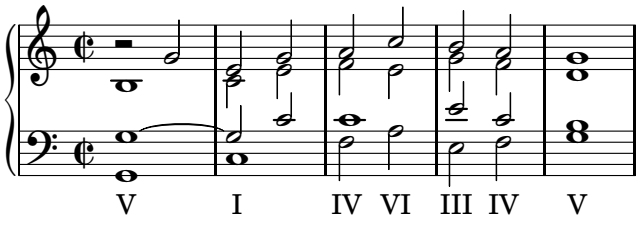

Figure 9. Harmonisation of a given part, as given by Piston and DeVoto (1991, example 117).

upward one octave, is called an *unstable* chord. In classical harmonisation it is preferred to start and end a harmonisation with a chord that is not inverted. The piece begins and ends on an unstable chord in second inversion, which again is correct when calculating a sequence that has the shortest distance, but it is too strict in the sense of classical harmonisation.

5.1.4 Melody C

Melody C and its generated harmonisation can be seen in Figure 7, and the experts' ratings in Figure 8. This harmonisation was well received by the experts.

The fourth bar sequence (*II - V*) is regarded as pleasant. The progression starting in bar 3 (*I - VI - II - V - I*) is considered to be technically correct, in the classical sense. Chords that are criticised and found not to fit the melody are the second chord of bar 2 (*III*), and the second chord of bar 3 (*VI*). *III* often serves as a mid-way point between *I* and *V*, something which is not happening here. The *VI* chord in the third bar is problematic because the third note in the bar, *A*, creates a dissonance with the chord, as explained in Section 5.1.3.

Expert III notes that the harmonies are not a part of a compositional idea, but are, in a sense, a good estimation for the moment they are used. This can again be attributed to the fact that FHARM does not take into account properties of the overall melody.

Although perceived as a more pleasant progression than melody B, problems with regard to the inversions of the chords are again noted, with too many parallel movements in the notes of the chords.

5.1.5 Overall results

Overall, melody A is found to have the harmonisation that works best. The harmonisation of melody C is also relatively appreciated by the experts, while that of melody B is leaves plenty of room for improvement.

It is interesting to look at the functional analysis trees of the harmonisation. In particular, the harmony for melody B gives rise to a simpler harmony tree than those for melodies A and C. However, care should be taken when determining this factor as a cause of the poor quality of the harmonisation, given the low number of melodies used in this experiment.

As expert III noted, one of the major elements that is missing from the harmonies is the presence of dissonant chords that are not the result of melodic jumps inside of a bar. This follows from picking three chords for each melody note, where each chord contains the melody note. This method will never yield dissonant chords with regard to the melody, something which is needed to make an "interesting" harmonisation, as consonances are regarded to be points of arrival, rest, and resolution. Without the use of dissonance, chord sequences can sound stale and mechanical.

5.2 Experiment two

In this experiment, a harmonisation example from Piston and DeVoto (1991, example 117), shown in Figure 9, is chosen as a ground-truth for FHARM. The manual harmonisation process of this melody by Piston is extensively documented in his book. We compare his choices to those made by FHARM.

5.2.1 Method

As before, FHARM generated one sequence for the melody (out of 1000 candidate sequences), as described in Section 4. The first melody note from Piston's melody was doubled as two half notes, as FHARM cannot currently handle melodies that do not begin on the first beat of the first bar.

5.2.2 Results

We now discuss the difference between the Piston's harmonisation and the one generated by FHARM. The first difference to notice is the placement of the chords, which differ in two ways. First, Piston used chords that are, in general, placed an octave higher. This way, some chord notes overlap with the melody notes. FHARM uses chords that are two octaves lower than the melody notes, preventing clashing of melody notes with chord notes, and making it easier to read and analyse the harmonisation. Even though it is harder to read, the approach taken by Piston sounds like the harmony blends in more with the melody, because the melody notes and chords are placed closer together.

The second difference is the spacing of the chord notes. In Piston's example, the first *V* chord shares the lowest note with the *V* chord in the FHARM harmonisation, but differs in the higher notes. Piston creates a converging sequence, by using a wide *V* chord at

	I	VII	III	III	I	VI	II	V	I	IV	IV	V	I	IV	I	Overall
I	1	1	2	4	2	3	2	2	2	3	2	2	3	3	2	2
II	1	1	1	5	2	1	1	1	1	4	2	2	1	1	1	2
III	4	4	2	4	2	3	3	2	4	3	3	5	4	3	2	3
Average	2	2	1.7	3	2	2.3	2	1.7	2.3	3.3	2.3	3	2.7	2.3	1.7	2.3

Figure 8. Melody C: ratings of the experts of the individual chords and overall progression on a scale from 1 to 5 (second and first questions of the survey, respectively; lower scores are better).

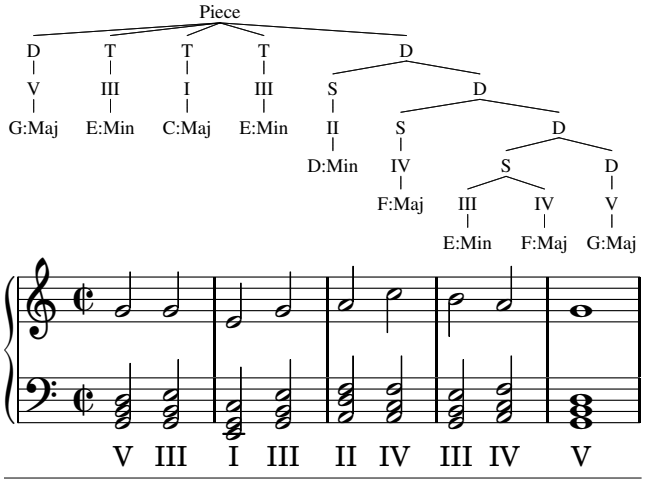

Figure 10. Melody of Walter Piston in C major with harmonisation (as generated by FHARM) and its harmonic analysis (as generated by HARMTRACE).

the start, and closing with a *V* whose notes are close together. Between those two chords, on average, the notes in the chords become closer. The convergence of notes in Piston's example creates the sense of a closing structure.

Because Piston takes into account chord voicing and sequence structure, his chord placement creates a harmony that sounds more classical. However, we cannot expect FHARM to "compete" with Piston in terms of voicing, as FHARM does not have such notion; it simply generates chords for a melody. Piston's harmonisation is entirely *polyphonic*, where each voice is melodious and independent. By contrast, FHARM generates a *melody-dominated homophony*, where the melody is clearly distinct from the accompanying voices, which support the harmony. Nevertheless, this changes little to the functional analysis on the individual chords and sequence, which is what we are really interested in comparing.

Both Piston and FHARM open the harmonisation on *V*. As there is no note on the first beat in Piston's example, it would be natural to harmonise the bar with the note on the second beat. This note is a *G*, which makes it a logical choice to select *V* for that bar, as the root of *V* is a *G*. FHARM had the choice between two chords on the first note *G*. This is because although *G* can be harmonised with *I*, *III*, and *V*, FHARM does not allow to begin a harmonisation with *III*, as explained in Section 4.2.1. This means FHARM can harmonise the note with a *I* or *V*. The choice for a *V* chord can be explained by its higher probability.

Although a *V* chord is more probable, FHARM harmonises the second *G* in the melody with a *III* chord. This creates a sequence with the following *I* chord in which all voices move a third down, in parallel. A better choice could have been made here, first because parallel movement is considered bad in classical harmony, and second because the combination of these chords do not create a functional entity, as can be seen from the lack of structure in the functional analysis tree pictured above the harmonisation.

The third melody tone is harmonised by Piston and FHARM with a *I* chord. Piston chooses to sustain this chord during the whole bar, whereas FHARM chooses a *III* chord on the second note (*G*) of the bar. Again, a *V* is more probable, but the probabilistic mechanism chose a *III* chord.

The *A* note on the third bar is harmonised by Piston with a *IV* chord, and by FHARM with a *II* chord. The choice of *II* by FHARM can be attributed to the probabilities of the possible chords, as the chord that Piston used, *IV*, has the lowest probability for the note *A*. The chord that is chosen, *II*, can be justified in a functional way, as it is part of a subdominant structure.

Both Piston and FHARM harmonise the first note of the fourth bar with a *III* chord. As can be seen from the structure of the tree, this *III* is part of a subdominant-dominant chain. A *III* is a good choice here, as it fits in a structure that can be functionally explained.

The last two chords, *IV* and *V*, are also identical in Piston's and FHARM's harmonisations. Piston used this sequence to create a *semicadence*, which is any cadence ending on *V*. It is called semicadence because it sounds incomplete or suspended. FHARM's choice of these chords can be explained by the method used in Section 4.2.1, which forces this cadence.

The algorithm for choosing chord inversions created subsequences containing parallel movement between the notes of the chords. This can be seen in the sequence *III–I–III* from bar 1 to bar 2, and *IV–III–IV* in bars 3 and 4. To prevent this, the *I* and *III* chord should be placed in different inversions. Despite this, the algorithm created a sequence starting and ending on a stable inversion, which is a good choice in the classical sense.

Overall, the generated sequence by FHARM bears a lot of similarity with the one by Piston. Although chords are chosen probabilistically, the choice of defining this probability in terms of distance in the circle of fifths, together with parsing of the harmonisation by HARMTRACE, results in a sequence that is similar to that of Piston.

5.3 Reflection on the evaluation

From the experiments carried out in this section, we learned that FHARM is capable of generating good harmonisations, but can be improved in several ways. Instead of only using one melody note to create a list of possible chords, the melody could be segmented into sections that are governed by a chord. As a result, the probability of choosing a chord could be described as a function from all the melody notes in that segment and the way the melody is evolving to the probability of a chord. To account for dissonant chords, the chord generation process should be extended to include chords that do not contain the current melody note.

Another improvement could be to take into account the phrase structure of the melody. Probabilities of chords could be increased or decreased depending on variation, strong or weak bar sections, and overall melodic structure. It would be interesting to see if we could analyse the melody first to derive information about its inner structure, and to look for variation and repetition. This

information is highly valuable for harmonising a melody in an interesting way, as variations and repetitions are usually reflected in the harmonisation.

Another possible direction for future research would be to investigate if the selection process of FHARM could be more tightly integrated with a HARMTRACE grammar. One way to achieve this could be using the melody to constrain parsing. Melodic information could be used in a similar way that key quality information is currently used as a constraint in the grammar. As can be seen from the production rules of a grammar in Section 4.3.1, creating a functional structure yields different results in a major or minor key. A similar approach could be used for the melody, on a per-note or per-phrase basis.

The current inversion process uses a trend line to choose the closest inversion of a chord to create the least varying harmonisation. Although this generally works well, and keeps the chords close to each other, our experiments have shown that, in some cases, the least varying inversion is not the best choice. To eliminate parallel and equal voice movements, larger chord spans should be considered, extending the three possibilities of inversions with notes that are close together. Using more information from the melody, better inversion choices can be made. For example, starting and ending on the unstable second inversion should not be allowed in opening and cadences of a sequence, but could be used on weak bar parts. Parallel octaves that are created between the upper voice of the chord and the melody should be avoided by choosing an inversion in which the upper note harmonises the melody note with an interval less or greater than an octave.

The process of selecting the best harmonisation for a melody uses a measure of fewest parser errors to determine the best chord sequence. From the experiments, we have seen that the worst harmonisation is the one with the least interesting functional tree. It would be interesting to see if a different measure of determining the best chord sequence could be derived from the inner structure of the functional tree. Having fewer nodes on the first level of the tree forces a more complex inner structure for the same number of chords. This way, the functional complexity of the harmonisation would be a measure of quality, as the number of nodes on the first level of its tree tells something about how chords can be explained with regard to their tonal context. As is shown from the tree structure of melody C, almost none of the chords share a functional identity with a neighbour. The only exception are chords that are repeated, but these do not yield an interesting progression.

6. Related work

Quite a few automatic harmonisation systems have been brought forward, with different approaches to the problem. In general, these systems can be categorised as rule-based models, statistical models, genetic models, and hybrid models. The last category includes hybrid models that use concepts of more than one type of model. In this section, we review a rule-based model, a statistical model, and a hybrid approached, and compare them to FHARM.

Rule-based approach Temperley (2004) proposed the knowledge-driven system "The Harmonic Analyser" that applies a Harmonic Preference Rule System to harmonise a melody. The model has two basic tasks: it must divide the piece into chord spans, and it must choose a root label for each chord span. After segmenting the melody, each of the possible 12 roots are assigned to each segment, which are given a score based on the weighted sum of the rules. The four harmonic preference rules focus on preferring certain tonal pitch-class root relations over others, preferring chord spans that start on strong beats of the meter, preferring roots that are close to the roots of nearby segments on the line of fifths, and preferring ornamental dissonances that are both closely followed by an event a step or half-step away in pitch height, and metrically weak. This approach is similar to FHARM, in that melody sections are used to select a chord. In FHARM, these sections are only one melody note long, and notes that are not assigned a chord are simply ignored in harmonisation. However, FHARM does not try to implement chord relationships within its rules, with the exception of forcing opening chords and cadence building. Instead, we solve this problem by processing the chord sequence through the HARMTRACE model, which selects chord sequences based on how correct their functional analysis is.

Quick and Hudak (2013) propose a grammar-based system for automated music composition. Like FHARM, their system is implemented in Haskell, but it generates both a melody and accompannying harmony. They solve the problem of voice leading using chord spaces (Quick and Hudak 2012), which would be interesting to apply to FHARM too.

Statistical approach A data-driven, statistical model that uses multiple hidden Markov models was introduced by Allan and Williams (2005). The authors describe how a dataset of chorale harmonisations by Johann Sebastian Bach can be used to train hidden Markov models. A probabilistic framework allows to create a harmonisation system which learns from examples, and can then compose new harmonisations. The framework of probabilistic influence allows performing efficient inference to generate new chorale harmonisations, avoiding the computational scaling problems suffered by constraint-based harmonisation systems.

Hybrid approach In a hybrid approach, Chuan and Chew (2007) proposed an Automatic Style Specific Accompaniment (ASSA) system, which, given only a few training examples, generates accompaniments in a particular style to a melody. This system uses statistical learning on top of a music-theoretic framework, therefore taking a hybrid approach. The relation between melodic notes and chord harmonies is modelled as a determination of a chord note. Notes are labelled as chord notes if they are part of a the chord structure, otherwise it is labelled a non-chord note. Each melody note is represented by 73 attributes, like pitch, duration, etc. With these attributes, the functional role of a melody tone can be described. Just as in Allan and Williams (2005), chord preference is learned from training examples. The resulting classifier is therefore determined by the style of the music in the training set.

In both of these approaches, all the harmonic rules are learned from training data presented to the models. In FHARM we use rules that are explicitly defined in a formal model. By changing this model, we can harmonise melodies in different styles, and even new styles that cannot be learned from any training set. Although building a new model in HARMTRACE can take some time, it allows for greater flexibility and precision with regard to the harmonisation rules.

7. Discussion and conclusion

We have presented FHARM, a chord generation and selection mechanism that uses the HARMTRACE model to automatically generate chords for a given melody. Our system begins by generating multiple candidate chord sequences based on the melody, and uses HARMTRACE to pick a sequence that best matches a harmony model. Unlike existing systems, FHARM creates chord sequences whose constituents are automatically functionally explained with regard to their tonal context.

FHARM should be seen as a first step towards an automatic, functional system for melody harmonisation. Its main contribution is the simplicity of the approach, and how it is built out of simple composition of individual components. Although the generated harmonisations leave room for improvement, we have seen that they are often good.

FHARM is a composition of four steps, as described in Section 4: generating chord sequences, selecting appropriate sequences, parsing to choose the best sequence, and post-processing. Improvements to FHARM could touch each of these components separately, without affecting the others. We could, for example, generate more chord sequences to begin with. We could also use different models in HARMTRACE while parsing, therefore preferring different styles of harmonisations. As for post-processing, we could use a different algorithm for picking chord inversions. All these improvements are specific to one component only, and require no changes to the remainder of FHARM.

Furthermore, FHARM relies on a number of key concepts central to functional programming: algebraic data structures for expressive data representation, parsing and pretty-printing, genericity and abstraction (in the use of HARMTRACE models), and functional composition. It is an example of how we can easily combine existing libraries (in our case HARMTRACE and HASKORE) together, while adding extra functionality on top. We hope FHARM can serve as an exploration tool for automatic harmonisation in Haskell, or as an inspiration for the development of other applications of functional modelling of harmony.

Acknowledgments

The readers of Koops (2012, upon which this article is based), Frans Wiering, and anonymous reviewers provided helpful feedback on an earlier version of this paper. José Pedro Magalhães is supported by EPSRC grant number EP/J010995/1. W. Bas de Haas is supported by the Netherlands Organisation for Scientific Research, NWO-VIDI grant 276-35-001.

References

M. Allan and C.K.I. Williams. Harmonising chorales by probabilistic inference. *Advances in Neural Information Processing Systems*, 17:25–32, 2005.

C.H. Chuan and E. Chew. A hybrid system for automatic generation of style-specific accompaniment. In *4th International Joint Workshop on Computational Creativity*, pages 57–64, 2007.

W.B. de Haas, J.P. Magalhães, R.C. Veltkamp, and F. Wiering. HarmTrace: Improving harmonic similarity estimation using functional harmony analysis. In *Proceedings of the 12th International Society for Music Information Retrieval Conference*, pages 67–72, 2011.

W.B. de Haas, J.P. Magalhães, and F. Wiering. Improving audio chord transcription by exploiting harmonic and metric knowledge. In *Proceedings of the 13th International Society for Music Information Retrieval Conference*, pages 295–300, 2012. ISBN 978-972-752-144-9.

W.B. de Haas, J.P. Magalhães, F. Wiering, and R.C. Veltkamp. HarmTrace: automatic functional harmonic analysis, 2013. Accepted for publication on the Computer Music Journal (37:4 Winter 2013).

C. Harte, M. Sandler, S. Abdallah, and E. Gómez. Symbolic representation of musical chords: A proposed syntax for text annotations. In *Proceedings of the 6th International Society for Music Information Retrieval Conference*, pages 66–71, 2005.

P. Hudak, T. Makucevich, S. Gadde, and B. Whong. Haskore music notation—an algebra of music. *Journal of Functional Programming*, 6(3):465–483, 1996. doi:10.1017/S0956796800001805.

H. V. Koops. A model based approach to automatic harmonization of a melody. Bachelor's thesis, Utrecht University, 2012.

S. Kostka, J.P. Clendinning, R. Ottman, and J. Phillips. *Tonal Harmony with an Introduction to Twentieth-Century Music*. McGraw-Hill, 2000.

J.P. Magalhães and W.B. de Haas. Functional modelling of musical harmony: an experience report. In *Proceeding of the 16th ACM SIGPLAN International Conference on Functional Programming*, pages 156–162. ACM, 2011. doi:10.1145/2034773.2034797.

S. Peyton Jones, editor. *Haskell 98, Language and Libraries. The Revised Report*. Cambridge University Press, 2003. Journal of Functional Programming Special Issue 13(1).

W. Piston and M. DeVoto. *Harmony*. Victor Gollancz, 1991.

D. Quick and P. Hudak. Computing with chord spaces. In *International Computer Music Conference*, September 2012.

D. Quick and P. Hudak. Grammar-based automated music composition in Haskell. *Proceedings of the 1st Workshop on Functional Art, Music, Modeling and Design*, 2013. doi:10.1145/2505341.2505345.

H. Riemann. *Vereinfachte Harmonielehre; oder, die Lehre von den tonalen Funktionen der Akkorde*. Augener, 1893.

J. Roeder. Pitch class. In *Oxford Music Online*. Oxford University Press, 2013. URL `http://www.oxfordmusiconline.com/subscriber/article/grove/music/21855`. Accessed May 22.

M. Rohrmeier. A generative grammar approach to diatonic harmonic structure. In Anagnostopoulou Georgaki, Kouroupetroglou, editor, *Proceedings of the 4th Sound and Music Computing Conference*, pages 97–100, 2007.

M. Rohrmeier. Towards a generative syntax of tonal harmony. *Journal of Mathematics and Music*, 5(1):35–53, 2011.

A Schönberg. *Theory of Harmony*. Univ of California Press, 1978.

T. Schrijvers, S. Peyton Jones, M. Sulzmann, and D. Vytiniotis. Complete and decidable type inference for GADTs. In *Proceedings of the 14th ACM SIGPLAN International Conference on Functional Programming*, pages 341–352. ACM, 2009. doi:10.1145/1596550.1596599.

S.D. Swierstra. Combinator parsing: A short tutorial. In *Language Engineering and Rigorous Software Development*, pages 252–300. Springer, 2009.

D. Temperley. *The Cognition of Basic Musical Structures*. The MIT Press, 2004.

H. Thielemann. Audio processing using Haskell. In Gianpaolo Evangelista and Italo Testa, editors, *DAFx: Conference on Digital Audio Effects*, pages 201–206, October 2004. ISBN 8890147903.

A. Whittall. Functional harmony. In *The Oxford Companion to Music*. Oxford University Press, 2013. URL `http://www.oxfordmusiconline.com/subscriber/article/opr/t114/e2730`. Accessed May 22.

A. Questions posed to the experts

1. Please rate the technical aspect of the total progression with a number from 1 to 5, where the rating indicates whether the sequence progresses according to the general rules of harmony theory, where:
   1: very correct; 2: moderately correct; 3: not correct nor incorrect; 4: incorrect; 5: very incorrect.
   a Could you explain this rating?
   b Please tell something about the creative aspect of the total progression. (Is it surprising, boring, mechanical, etc.)
   d Do you have any other remarks about the progression?
2. Please rate every chord and its melody segment individually with a number between 1 and 5, where the chord fits the notes:
   1: very well; 2: ok; 3: mediocre; 4: bad; 5: incomprehensible.
   a Could you clarify these ratings?
   b Which chords would you replace, and by what chords?
3. Do you have any other remarks about the harmonisation?