MusicHastie: field-based hierarchical music representation

Charles Fox
School of Computer Science, University of Lincoln, UK
chfox@lincoln.ac.uk

ABSTRACT

MusicHastie is a hierarchical music representation language designed for human and automated composition and for human and machine learning based music study and analysis. It represents and manipulates musical structure in a semantic form based on concepts from Schenkerian analysis, western European art music and popular music notations, electronica and some non-western forms such as modes and ragas. The representation is designed to model hierarchical musical perception by human musicians so can also be used to aid human understanding and memorization of popular music pieces. An open source MusicHastie to MIDI compiler is released as part of this publication, including capabilities for electronica MIDI control commands to model structures such as filter sweeps in addition to keys, chords, rhythms, patterns, and melodies.

1. INTRODUCTION

Humans usually listen to, play and create music as hierarchical structures. These range from low-level structures such as the composition of chords and rhythms, through intermediate structures such as chord progressions and melody phrases, to top-level structure such as sonata form or verse-chorus-verse types of popular song forms. However when we write and read western European art music, we still typically work only at the level of individual notes, with just a few of these higher levels of structure such as bars, keys and simple repeat symbols added to our note-based notation. During composition and analysis, we might make use of temporary notations which are discarded from our published sheet music. For example, analysts may annotate a score with the names of the chords at each beat or bar, first with absolute names such as G Major, and later with key-relative names such as 'V major, in the key of C'. Long-term key changes are usually notated in the sheet music, but temporary modulations might not be, but can be reconstructed and annotated by the analyst, such as 'modulate to the key of the dominant for one bar'. The analyst may then begin to chunk temporal sequences of chords into larger structures, such as the components of a sonata form or the lines and verses of a pop song. Schenkerian analysts [1] will also chunk the overall tonality of these components into a temporal hierarchy of modulations and sub-modulations. Some composers explicitly invert this process to generate compositions top-down [2].

Such chunking of music is thought to be what is performed by expert from-memory performers [3]. Humans find it difficult to remember long sequences of arbitrary symbols, but can store entire symphonies, fake books, or albums of data when they correctly chunk and hence deeply understand its structure. For example, the chords of the verse from the Beatles' *Hard Day's Night* might be understood by intermediate musicians as structures such as,

```
VERSE = X Y X Y Z X
X = C F C
Y = Bb C
Z = F  G
```

where each element of the verse, X, Y and Z is broken down into a short chord sequence, and the verse is viewed as a sequence of such elements.

A more expert musician will however have less work to do to perceive and remember this verse. This is because being an expert, within a particular musician culture, means being aware of a large repertoire of standard structures used within the culture. For example, the fragments above occur in many other songs which the musician already knows before *Hard Day's Night*, and the musician or their community might given them nicknames or formal names, such as *Trip, Bounce* and *Turn*,

```
Trip = i iv i
Bounce = viib i
Turn = iv v
```

Similarly, higher level structures such as verse forms may be well known and given standard names such as '*Ababca*',

```
ABABCA = X Y X Y Z X
```

For this expert, understanding, memorizing or composing *Hard Day's Night* becomes a much simpler task, because the new information required is now very compact:

```
VERSE = ABABCA
X = Trip
Y = Bounce
Z = Turn
```

Pop and jazz musicians thus tend to find that their personalised songbooks (fake books) become more compact over time, as they can shrink notations in this way. This is a reason why such musicians should maintain their own such songbooks rather than buying them off-the-shelf. Jazz real books [4] in particular are often sold including complex chord substitutions and un-intuitive keys which detract from understanding the deep structure of the songs. If the deep structure is understood then it becomes easy to create these substitutions and key movements on-the-fly.

A larger class of structures can be understood if the concept of *transforms* is added to such representations. For example, a common pop idiom ('standing up from the stools') can be represented as,

```
STOOLS=CHORUS VERSE CHORUS VERSE &+2.CHORUS
```

where the transform symbols before the final chorus here mean 'change key up by two scale degrees'.

The *Hard Day's Night* description above only describes the ordering of its chords, rather then their durations. If we introduce a time-stretch transform then we can capture the durations too by redefining,

```
TRIP = ~1/2.i ~1/2.iv i
```

where the new symbol represents the transform to stretch the following structure to half its length. (i.e. the fist two chords in *Trip* are played twice as fast as all the other chords in the verse).

Actualized music does not consist of chords such as I, IV, V or even their instantiations into a key such as C, F, G. Rather composers including improvisers draw particular notes from the current key-instantiated chord and it is these notes which are actually sounded and given to the listener. This can be done by sounding multiple notes at once, such as,

```
TRIAD = [1 3 5]
SEVENTH = [1 3 5 7]
```

where the square brackets here mean that their contents are to be played in parallel rather than in series, and the number symbols are the first, third etc. notes of the current chord in the current key.

Alternatively, the key and chord can be revealed [1] to the listener over time via a series of notes in a monophonic melody, such as,

```
RIFF = 1 1 ~2.5 3 3 7 1
```

where the fifth is here time-stretched to be twice as long as other notes. [2] Chords and melodies which include sevenths and other notes in addition to 1,3,5 can give more information about the key and chord. For example including a seventh can reveal if a chord is the tonic or dominant because these would give rise to different (major and flattened) seventh notes.

To express a composition to the level of actualised notes, we thus need to keep track of two generally independent and parallel temporal processes. First, the key and chord evolve over time, but are not directly observable. Second, actual monophonic (melodies) or polyphonic (instantiated chords) notes are drawn from these underlying key and chords (and later, other state parameters) according to temporal patterns such as the *TRIAD* and *RIFF* above. A rough analogy with physics may help to visualize this process: an evolving musical 'field' is like an unobservable quantum field, while the actualized notes drawn from it are like particle observations from it.

We have already introduced the square bracket notation for showing events occurring in parallel, so we can made us of it again to denote such parallel evolution of fields and melodic/rhythmic note draws, for example,

```
SONG = [ CHORDS +_.STRUMS ]
CHORDS = i v i
STRUMS = TRIAD TRIAD TRIAD
```

can denote three guitar-like strums played in sequence, overlaying a field which progresses through chords I, V, I. Each strum is a triad as previously defined. But each strum takes its pitch shift ('+') transform parameter from the field. Here '_' is a special symbol which tells its transform (an otherwise ordinary '+' pitch shift transform) to take its value from the underlying field at each point in time.

The notation introduced above is novel but is a modification of the previous MusicGenie [5] representation [3], which in turn was based upon the GCDL programming language [6]. GCDL was a full language in the sense being Turing-complete, developed for algorithmic composition [7]. MusicGenie simplified GCDL, losing Turing-completeness in order to crate a smaller L-system [8] based representation which was easier for manipulation by automated composition. However a smaller language mans larger programs are needed to express musical ideas than in a larger language, and MusicGenie programs were found to be quite unwieldy for its other originally intended use-case of human composition assistance for this reason.

MusicHastie, as used in the examples above has been developed to remedy some of the problems found by MusicGenie users. While automated composition has found niche applications such as real-time video game music generation conditioned on gameplay [9], the most interest in MusicGenie style representation has come from human composition, human and automated [10] analysis and performers, using it to aid their human understanding and memory of musical structure. Live coding such as algorave [11] has grown in importance and could form an additional application. But these are all fundamentally human applications which lead to different emphases in their requirements. [4]

Table 1. Formal grammar of MusicHastie

MUSICHASTIE → (LINE CR)+
LINE → NAME '=' (TERM)+
LINE → NAME '=' '[' (TERM)+ ']'
TERM → (TRANSFORM '.')* NAME ' '

2. SPECIFICATION

MusicHastie consists of one of more lines of definitions. Each line defines a name as equal to a right hand side collection of terms, to be played either in series or parallel. Each term is a name which may be prefixed by several transforms. The formal grammar for MusicHastic is shown in table 1, as standard regex description, where NAMEs are strings such as 'VERSE' and 'TRIAD' (but excluding the reserved names 'n' and '_'), CR is carriage return, and characters in quotes are literals. '+' means one or more and * means zero or more copies of the contents of the preceding bracket [5]. 'SONG' is a special name which must be defined and is taken to be the top level song.

2.1 Primitives

The musical field is a major new contribution of MusicHastie which was missing from MusicGenie. As introduced above, the field concept allows us to cleanly separate out the ontology of keys and chord progressions from that of rhythms, melodies and orchestration. This allow for songbook-like files of chord sheets to be human- readable, studyable and playable without interference from the other information.

[1] Raga music focuses on this 'showing' the key and scale.

[2] Similar notation is used by some jazz and funk horn players, sometimes with additional symbols such as dot or bars to show octaves. Raga melody fragments can be similarly notated using Sa-Pa symbols.

[3] MusicGenie audio demos:
www.youtube.com/playlist?list=PLjrJD5nSYNjoQebwV7TWPe83k7g2FjIO6

[4] MusicHastie retains hooks for MusicGenie style automation, intended for future use with Metropolis-Hastings sampling and the acceptance method of [12]. The name 'Hastie' is derived from this 'Hastings'.

[5] This syntax is made cleaner, more human readable, and easier to type than MusicGenie's. Definitions are made such as 'SONG = VERSE +2.6.CHORUS' rather than MusicGenie's more cumbersome 'SONG ->{VERSE , (~2)(6)CHORUS }'

It also allows chord progressions and arrangements to be transferred easily from one song to another.

Any L-system based representation needs to bottom out by having one or more predefined primitives. MusicGenie used only one, here called 'n' (previously NOTE) which is a default tonic crochet within the key of C Major. All other structures were constructed as transforms of 'n'. For example, $\sim$ *2.+3.$6.&5.n* is a Bb minim in G minor.

MusicHastie's ontology is instead based on two primitives. It retains n as its only way to actually produce sounds, as in MusicGenie. But it adds a second primitive, '_' which represents a default field, having the tonic chord in C Ionian major at medium volume, unity timestretch, and zero values of any timbre control parameters.

The chord symbols such as i, iv, v seen above are defined as transforms of the primitive field, such as,

```
iv = +4._
vi = +6._
```

Then to draw notes from the field, the method seen in the introduction is used. The degree shift transform, +, acts within scale degrees, so if the key of the field remains in C major then +6._ will shift any actualizing notes into A minor rather than A major.

2.2 Pitch transforms

The pitch transforms (first section of table 2) are the same as in MusicGenie, but can now act on either the field or on notes. Both the field and notes carry key, scale, and pitch information. For the field, these define chordality, for example C Major, degree 6 means A minor chord. For a note, they define its individual pitch, so C Major, degree 6 means the note A.

Transforms are provided to raise and lower the pitch degree (+,-) and to raise and lower the key (&) by degrees of the current state. For example &6._ raises the key from C major to A major. [6]

The mode transform $ rotates (in the sense of group theory) the scale pattern by i degrees, for example $2 rotates from Ionian to Dorian. In addition, a new scale select form is provided which sets the scale to Ionian major (M), harmonic minor (h), melodic minor (m) or extended (7 note) blues (B) scales. The nondiatonic scales and their many modes are unobtainable as modes of the default Ionian, but are important in musics such as Raga and Klezmer.

2.3 Timbral transforms

As in MusicGenie, transforms are provided to (a) multiply the velocity (roughly volume) of its target and (v) shift the MIDI channel of its target. In orchestral-like settings it is intended that MIDI channels should be arranged in some timbral order, such as the angular seating position of instruments in the orchestra, so that small and large shift of channel correspond to similar sized timbral shifts.

A new transform is now also introduced to shift MIDI control parameters, which is especially important as the foundation of electronica – especially trance – styles in which 303-like instruments' filters are shaped over time to create their 'hands in the air' effects. A major contribution

[6] An experimental version also provides similar but raising key i steps around the circle of fifths, which some musicians find more meaningful

Table 2. Transforms. i and j are integers, i/j are rationals, s, s_i and r are strings. e can be either integer or rational. Dagger=not in MusicGenie.

+i	up i scale degrees
-i	down i scale degrees
&i	key shift up i degrees
$+$i$	mode shift i degrees
$$s$	scale select ($s \in$ {M,h,m,B}) †
ae	modify amplitude by e
vi	change MIDI channel by i.
*i, j	change MIDI parameter i by j †
$\sim e$	time stretch by e
R	retrograde (play backwards)
ri	rotate sequence round by i terms †
f/s/	replace whole structure by vamps of s †
f/s_1,s_2,s_3/	as above with transition fills †
w(i, j)	swap the ith and jth terms †
s/s/r/	replace s terms with r †

of this work is the realisation that this can be achieved using exactly the same field system as previously introduced for other purposes – i.e. decoupling of chord sequences from rhythm and melody. By applying MIDI control shifts to the field rather than to notes, with operations such as *5,5._, their trajectories can similarly be decoupled from all other musical activity, allowing for their clean and separate specification. Actualized notes and note structures then simply need to take their control parameter settings from the field with operations such as *_.n.

2.4 Structural transforms

As in MusicGenie, transforms are provided which act by changing the structure of their targets. Timestretch ($\sim$) multiplies the duration of all notes within a structure by a rational, to stretch or shrink its total length. All notes types other than crochets are formed ultimately as timestretches of primitive n. Retrograde (R) flips a structure backwards.

A new sequence rotation operator is introduced (r) which moves the first i terms to the end (for example, 1,2,3,4,5,6 rotated by 2 is 3,4,5,6,1,2). A new swap operator (w) is introduced which swaps the ith and jth elements of a series, such as (1,2,3,4,5,6 becoming 1,2,5,4,3,6). These transforms are useful to represent algorithmic styles such as minimalism and change ringing.

Vamps and fills are introduced to model the thought processes of drummers, bassists, and tracker-style electronic musicians. These musicians wish to work from the same top-level song structure as chord and melody players (SONG = VERSE CHORUS VERSE) but interpret each of the high level components as an instruction to play a single repeated structure (vamp) for the whole duration of that structure. For example, f/ROCK1/.VERSE will $f$ill the whole of a VERSE with ROCK1 drum patterns.

An extension is introduced which can specify a set of patterns to use during the vamp, in increasing order of 'power'. MusicHastie does not specify exactly how a sequence should be chosen from them, but implementations of automated accompaniment systems may for example interpret f/ROCK1,FILL1,FILL2/.SONG as a series of increasingly powerful drum patterns and choose to deploy

the most powerful as fills to mark the largest boundaries in the structure, between verse and chorus.

Another new feature is representation *variations* of structures. For example, two VERSEs may both be instances of VERSE yet have different arrangements, melody lines, or solos as their parts. This is done via regex-like transforms such as s/STRUM/ARP/.VERSE describing a VERSE with its usual STRUM patterns *s*ubstituted by ARPs.

2.5 Imports

To model the concept of a cultural knowledge base (CKB), a library import system is introduced in MusicHastie. For example this allows common structures such as *Triad* and *Turn* above to be defined in libraries and imported by a new song which can then have a minimal description as for *Hard Day's Night* above. Imports are described by,

```
import <libraryname>
```

3. REFERENCE IMPLEMENTATION

`gitlab.com/charles.fox/musichastie` is an open-source Python reference implementation of a MusicHastie to MIDI compiler, together with a basic CKB, *Hard Day's Night* input, MIDI, and mp3 rendered examples. It uses UNIX 'everything is file' philosophy, taking a text file (and its imports) as input and outputting a MIDI file. It is run as *mh filename.mh*, outputting *filename.mid*. Unlike MusicGenie it does not include any GUI file editor or MIDI player, rather any external tools can be used. The compiler ignores blank lines and line contents after a "#" comment symbol. It requires the top level structure to be called SONG and crates the midi content from it. A basic interpretation of automated accompaniment is included which plays f/ROCK1,FILL1,FILL2/.SONG by searching the SONG for large and medium structural boundaries and deploying the fills at those locations.

3.1 Cultural Knowledge Base (CKB)

A CKB is provided as a set of library (.mh) files containing standard structures, currently based on Western pop. The following are short representative extracts.

chords.mh defines standard chords, as transforms on the field, such as the vi, vi, TRIAD, SEVENTH seen above.

notes.mh defines standard note types such as

```
m = ~2.n #minim
q = ~1/2.n #quaver
1 = +1.n #tonic crochet
2 = +2.n #supertonic crochet
1q=+1.q #shorthand, tonic quaver
2q=+2.q #supertonic quaver
3q=+3.q
1'q=+8.1q #octave up
2'q=+8.2q
_q = a0.q #quaver rest (zero amplitude)
```

progressions.mh defines both standard named chord progression fragments, such as Trip, Bounce and Turn above, and full progressions in terms of them, such as,

```
Basic = i iv
PlagalCadence = iv i
PhyrigianCadence = $+6.vii i
Amen = PlagalCadence #alternate name
Lift = i v
Sigh = vi iv
SensitiveFemale = Sigh Lift
AxisOfAwesome = Lift Sigh
PachelbelCanonLine2 = PlagalCadence Turn
```

forms.mh defines large-scale song forms such as *Ababca*.

arps.mh defines standard monophonic arpeggio and polyphonic strum patterns such as,

```
4Strums = TRIAD ///
DoubleWalk = ~1/2.Walk ~1/2.Walk
Walk = 1 3 5 3  #wealking bass
ArpPhilipGlass = 1 3 5 1' 5 3
ArpBachCello   = 1 5 3' 1' 3' 5  3' 5
```

drums.mh describes single and simultaneous drum hits in a novel 'tum-cha' notation, analogous to tabla note naming but for western pop kits. It uses these names to construct standard bar and multi-bar patterns such as,

```
UM = v9.-12.-7.n  #midi kick
AA = v9.-12.-3.n  #midi snare
TI = v9.-12.+3.n  #midi hihat
AM  = [AA UM]   #kick+snare combo
TUM = [TI UM]   #kick+hat combo
TA  = [TI AA]   #snare+hat combo
TAM = [TI AA UM] #kick+snare+hat combo
ROCK1 = TUM TI TAM TI
ROCK2 = ~1/2.TUM ~1/2.UM TI TAM TI
DNB = ~1/2.DNB_HALFSPEED    #drum-n-bass
DNB_HALFSPEED = TUM TI TA TI TI TUM TA TI
FILL2 = ~1/4.FILL2_QSPEED     #a big 1-bar fill
FILL2_QSPEED = TAM TA TA TA TA TA TA TA TUM TA TA TA TAM TAM TAM TAM
```

midi.mh describes standard trajectories for MIDI parameters, such as trance-style filter cutoff (parameter 74) sweeps,

```
1CutRise = *74,0.z  *74,2.z   *74,4.z  *74,6.z
z = ~1/2.a0/10.n
```

3.2 Full song

The chord sequence for the whole of *Hard Day's Night* can then be compactly notated as,

```
import chords, progressions
CHORDS = VERSE x2 CHORUS VERSE LINEA x2
VERSE = ABABCA
LINEA = Trip
LINEB = Bounce
LINEC = Turn
CHORUS = CLINEA CLINEB
CLINEA = vi  ii  vi /
CLINEB = Basic  v /
```

An entire Band-in-a-Box style arrangement of *Hard Day's Night* including drum patterns and fills, bass arpeggios, guitar strums, and cutoff sweeps (but omitting melody here, for space) can then be notated as,

```
import drums, forms, arps, midis
import harddaysnight.mh #easy to swap out
#import heyjude.mh  #easy to swap in
SONG =  ARRANGEMENT
ARRANGEMENT = [FIELD  +_.f/DoubleWalk.FIELD  v1.f/4CutRise.FIELD
               v1.+_.f/4Strums.FIELD f/DNB,FILL1,FILL2.FIELD ]
FIELD = ~4.CHORDS
```

4. REFERENCES

[1] T. Pankhurst, *SchenkerGUIDE: a brief handbook and website for Schenkerian analysis*. Routledge, 2008.

[2] D. McCabe, "David Cope: Techniques of the Contemporary Composer," *Computer Music Journal*, vol. 23, no. 4, pp. 100–102, 1999.

[3] R. I. Godøy, A. R. Jensenius, and K. Nymoen, "Chunking in music by coarticulation," *Acta Acustica united with Acustica*, vol. 96, no. 4, pp. 690–700, 2010.

[4] M. Mauch, S. Dixon, C. Harte *et al.*, "Discovering chord idioms through Beatles and Real Book songs," in *ISMIR*, 2007.

[5] C. Fox, "Genetic Hierarchical Music Structures." in *American Association for Artificial Intelligence AAAI-FLAIRS Conference*, 2006, pp. 243–247.

[6] S. Holtzman, *Generative Grammars and the Computer Aided Composition Music*. Ph.D. Dissertation, University of Edinburgh, 1980.

[7] J. D. Fernández and F. Vico, "AI methods in algorithmic composition: A comprehensive survey," *Journal of Artificial Intelligence Research*, vol. 48, pp. 513–582, 2013.

[8] P. Prusinkiewicz and A. Lindenmayer, *The algorithmic beauty of plants*. Springer-Verlag New York, 1990.

[9] C. Plut and P. Pasquier, "Generative music in video games: State of the art, challenges, and prospects," *Entertainment Computing*, vol. 33, p. 100337, 2020.

[10] C. Fox, "ThomCat: A Bayesian Blackboard Model of Hierarchical Temporal Perception," in *Proc. AAAI FLAIRS Conference*. AAAI, 2008, pp. 592–599.

[11] N. Collins and A. McLean, "Algorave: Live performance of algorithmic electronic dance music," in *Proceedings of the International Conference on New Interfaces for Musical Expression*, 2014, pp. 355–358.

[12] T. M. Browne and C. Fox, "Global Expectation-Violation as fitness function in evolutionary composition," in *Applications of Evolutionary Computing*. Springer Berlin Heidelberg, 2009, pp. 538–546.