Hooks Handbook

... a draft, as of 02 March 2013

In blue: Matthew Wilson own words.
In green: not yet implemented.
In brown: implementation details, for developers.

I. Principles

Purpose

There are several categories of music software. Sequencers and Digital Audio Workstations record, edit, mix, and play back digital audio, and are the tools primarily of songwriters creating a demo, recording artists, and producers. Scorewriting software packages, on the other hand, enable songwriters to input, edit and print music notation during and after the composition of music. Some sequencers include some scorewriting abilities; some scorewriters include some sequencing capabilities.

The overall purpose of Hooks is to enable a musician to quickly evaluate many melody candidates for fitness/quality as hooks for popular or orchestral music. The musician needs to efficiently manipulate aspects of a candidate to experiment with improving it. In order to do this, Hooks shares some features with scorewriting software and some sequencing features. It is able to generate multiple Hook-like phrases, and string them together to make a full song.

Main workflow

The user opens one or more databases, either already existent (its previous work), or new ones presenting a batch of mutated base hooks.

He/she explores variations of the hooks in those database, modifying their algorithmic structure, mutating them and eventually manually editing some of them. He/she groups interesting hooks together in a single database, and use them in songs structure. Eventually he/she manually edit the songs.

The source data is a set of two databases: "base hooks" and "mutable base hooks". The "base hooks" are supposed to be cloned and modified algorithmically, possibly to create new "mutable base hooks". The "mutable base hooks" are ready to be mutated in batch and give a set of children worth exploring.

Both databases can be opened from the main menu. Moreover, all "mutable base hooks" are listed in a submenu allowing the user to get a database of children of any of these hooks in a single gesture.

Collections (libraries) of resources

The interest of Hooks as an assistant for musical composition is the way it structures and makes available its musical knowledge.

- Base hooks are interesting by themselves: they have been choosen by the developers for their musical interest. Their algorithmic structure is transparent so it is easy to understand how they have been created and how they can be changed.

- Mutable base hooks are choosen for their fecundity; they are based on heavily random or large spectrum generators allowing a wealth of different interesting variations to be generated.

- Generators represent a set of compositional tools.

- Mutators represent strategies for modifying or replacing generators

- Metrics are ways to analyse a hook. They can guide hook mutation by selecting mutants.

Musical libraries (bank)
- Meters, so far one of 4/4, 2/2, 3/8, 6/8 or 9/8

- Keys
    major: C G D A E B F# C# Ab Eb Bb F
    minor: A E B F# C# G# D# A# F C D

- Chords
    - diatonic chords (auto)
    - close chords (auto)
    - borrowed chords (iio ii6 ii07 bIII bIII6 iv iv6 iv7 bVI bVI6 viio7 from minor), (I I6 from major)
    - other chords (V7sus4 I7 II II6 II7 III III6 III7 bVII bVII6 bVII7 VI VI6 VI7 VII VII6 VII7)

Song creation workflow

A song is created following these steps:
- algorithmic generation of a number of hooks
- oriented/controlled mutation and selection of some hooks
- manual editing of choosen hooks
- assembling of edited (or unedited) hooks according to a song formula
- manual edition of the song

II. Concepts & objects

Hook Structure

Algorithmic generation

A hook [class Hook] is based on a meter and a key; its length is specified as a number of bars. Within this framework it is algorithmically generated in layers, ordered as follow:

1 - harmonic rhythm
2 - harmonic progression
3 - chords rhythm
4 - actual chords (accompaniment)
5 - melodic rhythm
6 - melody (one or more voices)
7 - melodic ornamentation

Each layer is designed by a specific generation algorithm, called "generator" in the following.

When there are several melodic voices, they are generated in a definite order specific to the hook. Each new voice is processed by so-called "voice overlay rules" before it is mixed to the melody.

The tune generated by a hook is fully deterministic, even when its generators use random procedures: a given hook always generates the very same tune.

Mutation

Mutation is the process by which one or more generators is changed (a generator can be either altogether replaced by another one, or simply have some or all of its parameters changed).

When "in-place" mutation occurs, the hook is modified. In the case of "plain" mutation, a new hook is created, the mutated child of the initial hook. In-place mutation is then forbidden for the parent hook (see below).

A hook keeps track of its parent and children, if any. It also maintains a textual description of the mutation making it differ from its parent.

"Batch mutation" is applying a mutation several times in order to get a set of mutated children; it comes in two flavors: by default all children are generated by mutating the initial parent hook (they are brothers so to speak), while in cascade mode the mutation is applied in turn to each child (each mutated hook is parent of the next one).

When a hook has children it is said to become immutable: it is not possible to mutate it in-place any more (doing so would break the mutation relationship with its children and make the whole idea of hierarchical ordering of hooks meaningless). It is always possible to clone it and mutate the clone though.

Similarly, it is possible to clone an editable hook, be it fully editable or only harmonically editable.

Manual edition

A specific type of hook child [class EditedHook] can be manually edited: in such an editable hook the harmonic rhythm and progression can be interactively modified (using the editor in harmonic mode) and the melody arbitrarily tweaked (using the editor in melodic edition mode). Bars can be inserted or removed. Chords are still under the control of generators, which can be changed or modified in the same way as for a fully algorithmic hook.

Another specific type of hook child [class HarmonicallyEditedHook] can have its harmony only manually edited, while all other aspects of the tune are still controlled by generators. Such a hook can still be mutated; only the harmony is immutable.

Tune parts

A hook maintains a structured view of its tune. All rhythms used for generating progression changes, chords and melody are available for analysis or mutation, along with many other intermediate states.

Generators

All generators have an option (labelled "allow generic mutation" in their menu) allowing them to escape the effect of a generic mutation (see below).

1) Rhythmic Generators

Rhythmic generators work on so-called "base rhythms" which are the rhythms already defined in the hook when the generator is invoked; this may include meter and its variations (divisions, downbeats, on-beats, off-beats), harmonic rhythm and chordal rhythm: meter is always defined, harmonic rhythm is avaiblable for building chordal or melodic rhythm, chordal rhythm is only available for defining melodic rhythm. This simply follows the hook construction steps.

- The **basic rhythm** generator [class HGSimpleRhythm] follows one of the base rhythms, possibly making some beats void.

- The **pseudo-periodic rhythm** generator [class HGPseudoPeriodicRhythm] fragments one or more measures into smaller units then order them more or less differently over the meter.

- The **random rhythm** generator [class HGRandomRhythm] changes a base rhythm by either doubling or removing some of its beats, possibly making some beats void.

2) Harmonic Progression Generators

Progression generators may or may not handle chord voicing. Independently of their progression generation algorithm, they all have a "voice leading" setting which either leaves the progression as is, organize the chords octavic placement or compute a voice leading for smooth transitions [The implementation is rather naive at the moment, and under active development].

- The **simple progression** generator [class HGSimpleProgression] yields a predefined progression, either from an internal list or interactively from the user. It also define the harmonic rhythm (it is a multi-domain generator).

- The **cyclic progression** generator [class HGCyclicProgression] maintains a static chords progression and cycles through it along a Hook harmonic rhythm.

- The **expanded I ... V I** generator [class HGIVIProgression] expands the skeleton I-...-V-I progression into enough chords to fill the harmonic canvas.

3) Chords Generators

- The **strummed chords** generator [class HGChordStrummer] strums chords along the chordals rhythm non-void beats, according to a specific pattern.

- The **broken chords** generator [class HGChordBreaker] plays broken chords (arpeggios) along the chordal rhythm non-void beats.

4) Melodic Generators

All melodic generators have an articulation setting (specifying the articulation from staccato to fermata) and a register setting (three levels of transposition of the generated melody).

- The **random melody** generator [class HGRandomMelody] yields a non-repeating list of degrees of either the hook key or the underlying chord, in a specific range.

- The **rhythmic melody** generator [class HGRhythmicMelody] plays a different single note for each accent in the melodic rhythm.

- The **static melody** generator [class HGStaticMelody] maintains a static list of degrees (chordal, modal or chromatic) and cycles through them along a Hook melodic rythm, ignoring the void beats.

- The **arch-shaped melody** generator [class HGArchMelody] generates a melodic line from tonic to tonic with a single climax on a strong beat and a simple arch shape.

5) Ornamentation Generators

- The **dynamics** generator [class HGDynamicsComposer] leaves the melody unchanged, only tweaking notes amplitudes.

- The **progression appoggiaturas** generator [class HGProgressionAppoggiaturas] adds non-chord approaching tones prior to some notes located at harmonic change points.
It does so according to a rhythmic style dictionary telling how notes should be decomposed according to their rhythmic value. Different styles are available.
Weither the resolution is upward or downward, and stays so, depends on two parameters.
Grace notes are raised or lowered either modally (default) or chromatically.

- The **anticipations** generator [class HGAnticipations] adds anticipations to suitable notes (i.e. a note at harmonic change point belonging to current harmony, whose previous note does not belong to the current harmony but to the previous one).

- The **retardations** generator [class HGRetardations] adds retardations to suitable notes (same rules as for anticipation, plus a constraint that the previous note is one modal step away).

- The **passing tones** generator [class HGPassingTones] adds one to three tones linking suitable consecutive notes. It can be either chromatic or modal.

- The **neighbor tones** generator [class HGNeighborTones] adds a neighbor tone

between two consecutive notes of same pitch.

- The **neighbor groups** generator [class HGNeighborGroups] adds a double neighbor tones between two consecutive notes of same pitch.

The five above generators use a setting called "rhythmic mode" to decide where the newly generated tones are to be located. In each case, an existing note in the melody is being replaced by a group of shorter notes. That note time span is cut along the underlying meter or its subdivision, so that enough slots to place the new notes are obtained; when the rhythmic mode is #front, the first slots are populated; when #back we use the last ones; when #random, slots are taken at random.

- The **non-chord tones** generator [class HGNonChordTones] allows arbitrary combinations of the above generators. They can be individually enabled or disabled, and their settings are available.

Mutators

Mutators are responsible for modifying or replacing altogether some or all of a hook generators. In other words, they change a hook by modifying its algorithmic structure. They can either be applied in place, modifying the initial hook itself, or effect a child of the initial hook.

1) Generic Mutators

Generic mutators change the settings of a number of a hook generators; they do not replace any of the generators. Any generator can escape the effect of a genric mutator, if the "allow generic mutation" option available from its menu is turned off.

- The **seed change** mutator [class HMSeedChange] changes the random seed of its target generators

- The **incremental perturbation** mutator [class HMIncrementalPerturbation] incrementally and randomly modifies the parameters of its target generators (quantitative generic mutation)

- The **space explorer** mutator [class HMSpaceExplorer] randomly modifies some parameters and the random seeds of its target generators (vast quantitative generic mutation)

- The **modal perturbation** mutator [class HMModalPerturbation] randomly modifies the operational mode The "seed change" mutator [class

- The **global perturbation** mutator [class HMGlobalPerturbation] modifies all the parameters of its target generators , both quantitatively and qualitatively.

2) Specific Mutators

Specific (non-generic) mutators have an arbitrary effect. Few are implemented at the moment.

- The **melodic freezer** mutator [class HMMelodicFreezer] mutates a hook by hard-coding its melody and melodic rhythm. It replaces the current melody generator by a **static melody** one, and it replaces the melodic rhythm generator by a **hard rhythm**.

Metrics

Metrics provide different quantitative ways to analyse hooks structures. They can be used to filter mutated hooks via mutation sieves; they also provide immediate feedback when interactively editing a hook.

- **Harmonic tension** [class HMHarmonicTension] implements the following specification:

Another important detector is the one that detects harmonic consistency. See the Harmony document for the rules. Basically, the detector checks each chord transition (and starts/ends of phrases) for how well it follows the rules.

Built on the harmony detector is the tension detector. It identifies how often chord sequences leave us hanging, wanting them to resolve.

Built on those two detectors is one that determines whether resolution occurs on downbeats (for rock/pop: 1 and 3 over 2 and 4, 1 over 3).

count of exotic (non-diatonic) chords and melody
The number of times an exotic chord or note is used. One measure of exotic could be whether the chord or note is diatonic, that is, a note appearing in the scale of the key, or a chord appearing in the one of the 7 families of chords diatonic to the key.

- **Notes placement** [class HMNotesPlacement] details how notes populate specific beats and chord changes

- **Phrases** [class HMPhrases] implements the following specification:

Built on the phrase detector (HMRhythmicStructure) and harmony detector (HMHarmonicTension) is one that determines whether a phrase resolves to tonic.

level of unresolved tension :
Determined by the duration dominant-family chords are in use in the phrase, especially if they move to other dominant or subdominant chords (not tonic). There are other measures that could be used too.

number of transitional notes :
Determined by how many times a transitional note is used. A transitional note doesn't appear in the harmony chord playing during the note. There can be a chord change before or after the transitional note, or not at all. Usually the transitional note appears in the scale of the key, and is between its two neighbors in pitch.

number of chord changes :
Determined by the number of times the harmony changes chords. A phrase with a lot of chord changes is difficult to perform and probably doesn't sound very good (Exception: Phantom of the Opera 16th-note changes). A phrase with few chord changes is probably too monotonous. This algorithm is used when selecting which chord should go next in the chord progression - whether to stay on the same chord for that beat or to change chords.

size of melodic range :
The range of pitches the melody covers. Usually limit to two octaves. If the range is too small, the melody probably sounds too monotonous.

- **Rhythmic structure** [class HMRhythmicStructure] implements the following specification:

One of the (if not the) most important feature detectors is the one that detects repetition of

rhythmic structure. Strong repetition of a unique rhythmic pattern is assumed to greatly improve the musicality of popular music and some types of orchestral music. A repetition is defined as a set of equal-length phrases that include notes on 85(?)-100% of the same beats. Generally, at every point in the song, the algorithm compares all non-overlapping pairs of phrase segments with equal time duration, at eighth(?)-note starting point increments and bar-long duration increments (until half the song's duration is reached or some other reasonable maximum (20 seconds?), and keeps a count, length, and locations of each repetition set found. If a repetition is 8(?) bars or more and occurs at least 3(?) times, it might be categorized as a verse, especially if it contains its own set of rhythmic repetitions. Strengthening rhythmic repetition is melodic repetition. If a rhythmic repetition set's melodic contour is similar (identical on the intervals between their shared beats up to some threshold), it greatly increases its musicality, and heightens expectations that future pattern prefixes will continue/end similarly to how the previous repetitions did.

Built on that is a phrase detector, that detects lengths of phrases by intervening long-held notes or rests.

- **Rhythmic tension** [class HMRhythmicTension]

Syncopation measurement, rhythmic density analysis

accents (agogic, dynamic, tonic)

- **Voice leading** [class HMVoiceLeading] implements interval-wise metrics for a monophonic line

- **Voicing analysis** [class HMVoicingAnalysis] implements voice-leading metrics for polyphonic melodies

- **Melodic analysis** [class HMMelodicAnalysis] implements metrics dealing with tonality, pitch range and melodic contour

- **Chords** [class HMChords] lists all chords eventually making up the hook tune

Databases

Databases are collections of hooks, automatically including their children (i.e. their mutated variations).

Databases can also include one or more song formulas.

Each database maintains some parameters defining its behavior:
- weither its hooks are protected from deletion
- weither mutation should automatically happens in a new database
- weither batch mutation should occur in cascade
- the number of mutants in a batch mutation

A set of hard-coded databases is provided access via the main Hooks menu:

- **Base hooks** contains a set of hooks suitable for study and simple mutation

- **Mutable base hooks** contains a set of hooks suitable for batch mutation

- **Good tunes** (initially empty) is the database where the user is supposed to store the best hooks he/she comes across.

- **Interesting tunes** (initially empty) is for not-so-good but promising tunes.

- **In the works** (initially empty) is dedicated to current exploration

The user can register any database of its own to the Hooks menu.

Songs

A song is initially generated within a database by a song formula using the bindings current in the database.

A song formula editor make it possible to enrich the formula with effects applied to its parts.

A generated song can be further edited manually in a song editor.

III. GUI & editors

Database Editor (main GUI)

Database-wide settings are available from the editor menu, under the upper left blue round icon with a triangle:
- a toggle sets weither mutation should happens in a new database
- a toggle tells if batch mutation should occur in cascade
- the number of mutants in a batch mutation can be set
- the batch mutation timeout can be set
- "confirm hook deletion" defines weither the hooks are protected from accidental deletion (pressing the wrong button...)

The database editor is currently divided in four parts:

- in the bottom left window is the tree view of the database hooks

- in the upper window is the currently selected hook editor; its functionalities differ depending on weither the selected hook is manually editable or fully algorithmic; if, instead of a hook, a song is selected then the upper window displays the corresponding song formula editor

- below is a keyboard display

- at the bottom right is a text display.

In all editors contextual menus are triggered by a right mouse click.

Database Tree Editor

The tree view displays all the hooks in the database according to their inheritance: children icons are placed right below their parent icon, slightly offet to the right.

An immutable hook has its icon marked with a top-rigth blue triangle; an editable hook is marked with a similar golden triangle; a harmonically editable hook with a green one. All other hooks are mutable fully algorithmic hooks.

At any given time a single hook is selected; its icon is highlighted in gold. The tree editor contextual menu refers to that hook.

From that menu it is possible to rename the hook, edit its associated comments, play it, bind it to a letter for reference in a song formula, mark it as a favorite, clone it, move or copy it to another database, and eventually delete it, possibly with its children.

If it is also possible to get an editable version of the hook:
- when the hook is algorithmic:
    "edit" creates a fully editable child
    "edit harmony" creates a harmonically editable child
- when the hook is harmonically editable:
    "edit fully" creates a fully editable clone

Hook editors

1) Common features to both editors

The editors are very similar and share most of their features.

The manual editor is an instance of HookEditor class of which the algorithmic editor is a specialized subclass, HookAlgorithmicEditor. HookEditor is itself a subclass of the generic HooksMusicalPhraseEditor (specializing the muO MusicalPhraseEditor for Hooks) where the integration with a Hook instance and the overall look-and-feel is implemented.

Concepts

The window is the part of the score that is displayed. The user can scroll the window in all four directions by dragging the mouse (the cursor is displayed as a hand). If the <shift> key is pressed while dragging, the window is resized: this works as if the bottom left corner was pinned down and the window an infinitely extensive piece of rubber band.

It is possible to lock x and y

Two formats are available for chord names wherever they are written down in the editor: explicitly (tonic followed by the chord) and in roman numeral notation. This can be toggled from an item labelled "use roman numerals for chords" in the editor menu.

Playback

The hook tune can be played in several ways. The "play" submenu proposes the playback of the whole tune, the melodic part only or the chords only.

The "orchestration" submenu maps a MIDI instrument to each melodic voices, and one to harmonic accompaniment.

Key bindings:

?	Display key bindings summary
f	Resize window in order to show all music

Display:

The melodic notes have a specific color for each voice; their opacity reflect the notes volumes (rest are drawn as empty circles);their extensions are displayed in the form of semi-opaque rectangles.

Chord names are drawn at the bottom of the editor (they do not move vertically)

Optional displays:

The display of notes names can be toggled from item "display notes names" in the editor menu.

The notes harmonic status can be toggled via item "display notes harmonic status":

- a note tightly circled in green is a chord tone
- a larger green circle indicates the note is modal but a non-chord tone
- a large red circle means the note is non modal
- exceptionally a note can be a chord tone, while non modal (if the chord does not match the key); in that case is is circled in red and green

The metrics selected in the database editor text display are also responsible for their own layer of representation in the hook editor; see the chapter on metrics for details.

2) Algorithmic Editor

A hook is either mutable (when it does not have children) or immutable (if it has children):

- an immutable hook cannot be edited by itself, only indirectly: either via a new child (items "edit" or "mutate" in the editor menu), or as a childless copy ("clone" item).

- a mutable hook can have all the components of its algorithmic structure modified: meter, key, number and generation ordering of voices, number of bars, and generators.

a) Mutation

The mutation submenus gives access to several way to mutate the hook.

A fundamental mutation is simply generator replacement: selecting a new generator for any layer of hook composition. This occupy the first part of a mutation menu.

The second part is dedicated to generic mutators (see the chapter on mutators for details); it is itself composed of submenus: for each subpart of the tune to be effected, one can choose the type of generic mutator to be applied. For example, selecting item "change random seeds" under the "all rhythmic generators" submenu will have the three current rhythmic generators (for harmony, chords and melody) use new random numbers for their stochastic operations.

For a mutable hook, there is two mutations menus: "mutate" and "mutate in place". The first one operate on one or mode children of the selected hook (batch mutation). The second one applies the mutation on the hook itself. An immutable hook only proposes the "mutate" menu.

For batch mutation the number of children to be created is defined in the database (default is 10); see above the chapter on the database editor for details. Note that depending on the specifics of the mutation settings, is is not always possible to obtain the required number of mutants, either because there are not that many different mutants available (mutants are guaranteed to all output a different tune) or because the defined constraints (mutation sieves) filter out too many of them. Batch mutation is stopped after a database-dependent timeout (by default 2 seconds).

The "mutate" mutation menu has one extra item, "neutral mutation", which is simply the way to get a non-mutated child of the current hook. If the current hook is harmonically editable, this will yield a fully algorithmic clone.

b) Generators settings

The generator settings take the bottom part of the editor menu. Each generator has its own submenu, whose label starts with a two-letters code identifying the tune part composed by the generator:

HR	harmonic rhythm
HC	harmonic progression
CR	chords rhythm
CC	chordal accompaniment
MR	melodic rhythm
MC	melody (MC1, MC2, ... if several voices)
OC	melodic ornamentation

The generator submenu gives access to specific parameters and is different from each generator; see the chapter on generators for details.

3) Manual Editor

a) Harmonic mode

Clicking anywhere in the background, or on an harmonic note, toggles the editor into harmonic mode.

In harmonic mode the user can edit the harmonic progression and harmonic rhythm. At any time a chord region is selected, and a chord list is active for interactive cycling. The available chord lists are:

- "diatonic chords": the triads and seventh chords of the hook key
- "borrowed chords": the chords borrowed from the parallel key
- "other chords"
- "transition chords"
- "close (to previous) chord"
- "close (to next) chord"

The chord region can be split (for insertion of a new chord), deleted or have its duration changed. When a chord is deleted its region is appended to the previous chord (and consequently the first chord can not be deleted); when its duration is expanded the next chord region is reduced accordingly.

Display:

The selected chord region is highlighted in grey.

Keyboard:

The keys are painted in a way to reflect the underlying harmony: chord notes have their keys displayed in green, their octavic transpositions are painted in a paler green. Non-modal notes appear in dark grey; in case a chord note is non-modal, its green color is dulled.

The keyboard has no editing action in harmonic mode.

Key bindings:

tab	Change editing mode (enter melodic edition)
arrow left	Select previous chord
arrow right	Select next chord
space	Play selected chord
<alt> space	Play selected region (chord + melody)
arrow up	Cycle selected chord up within current chord list
arrow down	Cycle selected chord down within current chord list
<alt> arrow up	Cycle selected chord up within borrowed chords list
<alt> arrow down	Cycle selected chord down within borrowed chords list
<shift> arrow up	Cycle selected chord up within transition chords list
<shift> arrow down	Cycle selected chord down within transition chords list
+	Transpose selected chord one octave up
-	Transpose selected chord one octave down
i	Cycle over selected chord inversions
c	Split selected chord region in two
d	Delete selected chord

<alt> arrow left	Retract selected chord region (at right)
<alt> arrow right	Expand selected chord region (at right)
<alt> z	Undo last command

b) Melodic mode

Clicking any melodic note toggles the editor into harmonic mode.

In melodic mode the user can edit the melodic notes. At any time a voice is active and a note in that voice is selected. A note can be moved, transposed, transformed into a rest or be deleted; its volume can be changed. New notes can be inserted.

Some operations involve so-callled "ghost notes"; these are temporary placeholders marking empty spots available for moving an existing note or creating a new one. They are sperated by a sixteenth note value and fill the region around the selected note, up to the previous and next note in the voice.

Display:

The selected note features a golden outer ring; a vertical red line marks its location.

Ghost notes appear as smaller rest notes (technically they are rest notes).

Keyboard:

The selected note appears as a larger orange dot in the corresponding key. Its octavic transpositions appear as pinkish similar dots. The keys are painted in a way to reflect the underlying harmony (see above).

Clicking a key transposes the selected note to that key, and plays it.

Key bindings:

tab	Change editing mode (enter harmonic edition)
arrow left	Selected previous note
arrow right	Selected next note
space	Play selected note
arrow up	Transpose selected note one semitone up
arrow down	Transpose selected note one semitone down
<alt> arrow left	Move selected note a sixteenth left
<alt> arrow right	Move selected note a sixteenth right
+	Augment selected note volume
-	Diminish selected note volume
d	Make the note a rest; if it is a rest already, delete it
g	Display ghost notes around selected note
n	Instanciate selected ghost note or rest
<alt> g	Hide ghost notes

<alt> z Undo last command

c) Bars edition

In harmonic mode, when the selected chord region ends at a bar, it is possible to insert one or more measures via the "insert at bar n" submenu in the editor menu. [This is a limited functionality at the moment]

Song Formula Editor

...list and document commands here

Song Editor

...list and document commands here

IV. Real-time usage

Not implemented yet

V. Extension

Adding a generator

Blah

Adding a mutator

Blah

Adding a metrics

Blah

Adding a meter

Blah

Adding a song formula effect

Blah