Computing for Musicology (Course code: F104N5) 5. Towards Pattern Recognition in Music

J.N. Oliveira

Dept. Informática,
Universidade do Minho
Braga, Portugal

June 2009 (last update: May 2018)

Licenciatura em Música
(http://www.musica.ilch.uminho.pt/)
Universidade do Minho
Braga

What is a pattern?

From the Wikipedia:

> *A* **pattern** *(...) is a type of theme of recurring events or objects, sometimes referred to as elements of a set. These elements repeat in a predictable manner. (...)*

Still Wikipedia:

> **Pattern matching** *is the act of checking for the presence of the constituents of a pattern, whereas the detecting for underlying patterns is referred to as* **pattern recognition**.

Normally, queries involving *map*s and *filter*s extract information (eg. by counting) ignoring the patterns which layout such information.

Example

- Suppose we want to check whether a particular data element d **occurs** in a list l.
- There are several ways to provide an answer to such a **query**.
- The easiest is to evaluate $d \in l$ — the answer is a Boolean (*True* or *False*), with maximal loss of information.
- Another is to **count** the number of occurrences of d in l:

  $$check\ d\ l = (length \circ filter\ (\equiv d))\ l$$

  There is more information now — should d occur in l, we know how often.
- Still we have lost the information of where in the list such occurrences take place: all at the front? scattered? all at the tail?

Finding indices in sequences

The following function tells which **positions** in a sequence s are occupied with data satisfying a particular condition p:

$$findIndices\ p\ s = [i \mid (x, i) \leftarrow zip\ s\ [0\ ..], p\ x]$$

To see how *findIndices* is more informative than *filter*, run the following query inspecting “rondo word” `"ARBRCRBRA"`

$$findIndices\ (\equiv \texttt{'R'})\ \texttt{"ARBRCRBRA"} = [1, 3, 5, 7]$$

and compare with

$$filter\ (\equiv \texttt{'R'})\ \texttt{"ARBRCRBRA"} = \texttt{"RRRR"}$$

How *findIndices* works

1st step — zipping: zip `"ARBRCRBRA"` $[0\,..]$ yields

$$[(\texttt{'A'},0),(\texttt{'R'},1),(\texttt{'B'},2),(\texttt{'R'},3),(\texttt{'C'},4),(\texttt{'R'},5),(\texttt{'B'},6),(\texttt{'R'},7),(\texttt{'}$$

2nd step — filtering via $x \equiv \texttt{'R'}$ yields

$$[(\texttt{'R'},1),(\texttt{'R'},3),(\texttt{'R'},5),(\texttt{'R'},7)]$$

3rd step — selecting the right component i of each pairs (x, i), yielding

$$[1,3,5,7]$$

Word (sequence) inversion

Note how easy it is to record the sequence of positions occupied by all elements in a sequence:

$$invert\ s = nub\ [(x, findIndices\ (\equiv x)\ s) \mid x \leftarrow s]$$

For instance,

$$invert\ \texttt{"ARBRCRBRA"} =$$
$$[(\texttt{'A'}, [0, 8]), (\texttt{'R'}, [1, 3, 5, 7]), (\texttt{'B'}, [2, 6]), (\texttt{'C'}, [4])]$$

clearly tells the role of A (begin $=$ end), refrain R, intermediate episode B and middle episode C.

Searching for patterns

Let us now generalize

$$\mathit{isPrefixOf}\ \ p\ l$$

so that it checks whether a particular pattern p occurs in a list l at position i:

$$\mathit{match}\ p\ l\ i = p\ \text{`}\mathit{isPrefixOf}\text{`}\ (\mathit{drop}\ i\ l)$$

For instance, not only $\mathit{isPrefixOf}$ `"Mendel"` `"Mendelssohn"` $=$ True holds, but also

$$\mathit{match}\ \texttt{"ssohn"}\ \texttt{"Mendelssohn"}\ 6 = \mathit{True}$$

Clearly,

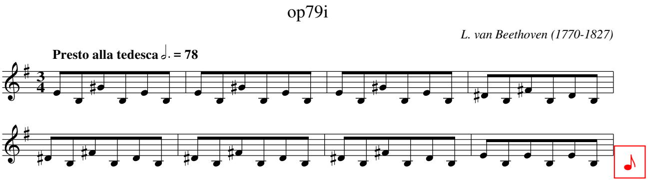
$$\mathit{isPrefixOf}\ \ p\ l = \mathit{match}\ p\ l\ 0$$

Searching for patterns

Last but not least, we may think of a function which records in which positions in a sequence a particular pattern occurs:

$$\mathit{patternIndices}\ p\ s = \\ [(i, i + \mathit{length}\ p - 1) \mid (x, i) \leftarrow \mathit{zip}\ s\ [0\,..],\ \mathit{match}\ p\ s\ i]$$

Consider, for instance,



Searching for patterns

Clearly, this piano sonata fragment (right hand only) is captured by

$$\mathit{tune} = \mathit{ntimes}\ \mathit{cell1}\ 3 + (\mathit{ntimes}\ \mathit{cell2}\ 4) + \mathit{cell3}$$

where

$$\begin{aligned}
\mathit{cell1} &= [\texttt{"E"}, \texttt{"B,"}, \texttt{"\^{}G"}, \texttt{"B,"}, \texttt{"E"}, \texttt{"B,"}] \\
\mathit{cell2} &= [\texttt{"\^{}D"}, \texttt{"B,"}, \texttt{"\^{}F"}, \texttt{"B,"}, \texttt{"\^{}D"}, \texttt{"B,"}] \\
\mathit{cell3} &= [\texttt{"E"}, \texttt{"B,"}, \texttt{"E"}, \texttt{"B,"}, \texttt{"E"}, \texttt{"B,"}]
\end{aligned}$$

So,

$$\begin{aligned}
&\mathit{patternIndices}\ \mathit{cell1}\ \mathit{tune} = [(0,5),(6,11),(12,17)] \\
&\mathit{patternIndices}\ \mathit{cell2}\ \mathit{tune} = \\
&\qquad\qquad [(18,23),(24,29),(30,35),(36,41)] \\
&\mathit{patternIndices}\ \mathit{cell3}\ \mathit{tune} = [(42,47)]
\end{aligned}$$

as expected.

Searching for patterns

However,

- One has the feeling that there is **only one** cell in this fragment which repeats at different degrees of the scale. Howe can we capture this?
- We need an **abstraction** mechanism which should be able to abstract from each cell the pattern of intervals involved.
- For this we need to model the notion of **interval** between two degrees in a diatonic scale.

Prior to all this, let us investigate how some other *music abstraction* functions can be encoded in Haskell.

More subtle filtering functionality

Think of the function *copy* which copies its input faithfully to the output, that is, $copy\ x = x$. Surely, this function has the following properties,

$$\begin{aligned} &copy\ [\,] = [\,] \\ &copy\ [x] = [x] \\ &copy\ (s + r) = (copy\ s) + (copy\ r) \end{aligned}$$

from which we easily calculate

$$\begin{aligned} &copy\ [\,] = [\,] \\ &copy\ [x] = [x] \\ &copy\ (x : r) = x : (copy\ r) \end{aligned}$$

as earlier on.

More about filtering

Function *copy* can be easily converted into another,

ndcopy (= "no duplicate copy")

that removes duplicates by adding a filter at each stage:

$$
\begin{aligned}
&\mathit{ndcopy}\ [\,] = [\,] \\
&\mathit{ndcopy}\ [x] = [x] \\
&\mathit{ndcopy}\ (x : r) = x : (\mathit{filter}\ (\neq x)\ (\mathit{ndcopy}\ r))
\end{aligned}
$$

NB: *ndcopy* is nothing but the standard function *nub* to which we have resorted earlier on.

More about filtering

- Between these two extremes (copying everything or removing all duplicates) there is the intermediate operation which removes only consecutive duplicates.
- To see the difference, compare

  $$ndcopy\ \texttt{"Mendelssohn"} = \texttt{"Mendlsoh"}$$

  (all duplicates go out) with

  $$ncdcopy\ \texttt{"Mendelssohn"} = \texttt{"Mendelsohn"}$$

  (only `"s"` in `"ss"` gets filtered).
- How do we encode *ncdcopy*?

Abstraction: removing local repeats

Removing **all** duplicates:

$$\begin{aligned}
&\mathit{ndcopy}\ [\,] = [\,] \\
&\mathit{ndcopy}\ [x] = [x] \\
&\mathit{ndcopy}\ (x : r) = x : (\mathit{filter}\ (\not\equiv x)\ (\mathit{ndcopy}\ r))
\end{aligned}$$

Removing **consecutive** duplicates only:

$$\begin{aligned}
&\mathit{ncdcopy}\ [\,] = [\,] \\
&\mathit{ncdcopy}\ [x] = [x] \\
&\mathit{ncdcopy}\ (x : y : r) \\
&\quad \mid x \equiv y = \mathit{ncdcopy}\ (x : r) \\
&\quad \mid x \not\equiv y = x : \mathit{ncdcopy}\ (y : r)
\end{aligned}$$

Removing locally repeated notes

Recall that music notes are pairs (n, d) of note **pitch** with **duration**.

Abstracting from **repeated notes** is trickier because we want to keep durations of the notes we are going to remove:

$$
\begin{array}{l}
nrep\ [\,] = [\,] \\
nrep\ [a] = [a] \\
nrep\ ((n, d) : (n', d') : l) \\
\quad \mid n \equiv n' = nrep\ ((n, d + d') : l) \\
\quad \mid n \not\equiv n' = (n, d) : nrep\ ((n', d') : l)
\end{array}
$$

Removing locally repeated notes

Consider, for instance, the beginning of the *Presto* of Beethoven's String Quartet op.74:

etc

Removing locally repeated notes

Compare the original part of the 1st violin,

with the same once subject to *nrep*:

(Note the binary meter flavour of the first bars, which could be thought of as being $\frac{6}{8}$.)

Removing locally repeated notes

In Haskell, here is (the beginning) of the original tune:

$\mathit{tune} =$
$[(\texttt{"c"}, \frac{1}{8}), (\texttt{"c"}, \frac{1}{8}), (\texttt{"c"}, \frac{1}{8}), (\texttt{"C"}, \frac{3}{8}), (\texttt{"e"}, \frac{1}{8}), (\texttt{"e"}, \frac{1}{8}), (\texttt{"e"}, \frac{1}{8}), (\texttt{"E"}, \frac{3}{8}),$

Now the effect of *nrep*:

$\mathit{nrep}\ \mathit{tune} =$
$[(\texttt{"c"}, \frac{3}{8}), (\texttt{"C"}, \frac{3}{8}), (\texttt{"e"}, \frac{3}{8}), (\texttt{"E"}, \frac{3}{8}), (\texttt{"g"}, \frac{3}{8}), (\texttt{"c"}, \frac{1}{4}), (\texttt{"e'"}, \frac{1}{4}), (\texttt{"c"}, \frac{1}{4})$

Sampling for musical analysis

- Removing repeated notes provides for music *abstraction* wherever rhythm is unimportant and tune (pitch) analysis is at target
- Quite often one wishes to *abstract* from the details of the tune itself and focus on the **tonal thread** by removing eg. *passing notes*, *grace notes*, and so on.
- **Sampling** does this for us, as shown next.

Sampling for musical analysis

In this case, a list of durations is the additional input (sampler) which tells at which points in time notes are to be selected, while keeping the durations specified by the sampler:

$$
\begin{array}{l}
\mathit{sample} :: (\mathit{Ord}\ d, \mathit{Num}\ d) \Rightarrow [d] \rightarrow [(n,d)] \rightarrow [(n,d)] \\
\mathit{sample}\ [\,]\ \_ = [\,] \\
\mathit{sample}\ \_\ [\,] = [\,] \\
\mathit{sample}\ (y:r)\ ((a,x):t) \\
\quad \begin{array}{ll}
\mid y > 0 \wedge y \equiv x & = (a,y) : \mathit{sample}\ r\ t \\
\mid y > 0 \wedge y < x & = (a,y) : \mathit{sample}\ r\ ((a,x-y):t) \\
\mid y > 0 \wedge y > x & = (a,y) : \mathit{sample}\ ((x-y):r)\ t \\
\mid y < 0 \wedge x + y \equiv 0 & = \mathit{sample}\ r\ t \\
\mid y < 0 \wedge x + y > 0 & = \mathit{sample}\ r\ ((a,x+y):t) \\
\mid y < 0 \wedge x + y < 0 & = \mathit{sample}\ ((x+y):r)\ t
\end{array}
\end{array}
$$

Sampling for musical analysis

Example: two different samples of op.74iii,

and

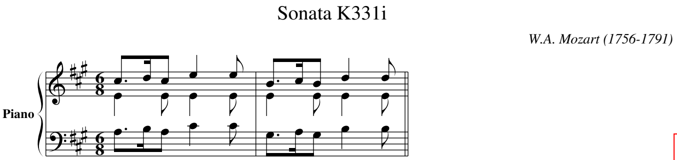
where the latter loses more information, keeping only the tonal thread.

Exercise 1: Write in Haskell the sampler lists which yield the above two samples of op.74iii main theme.

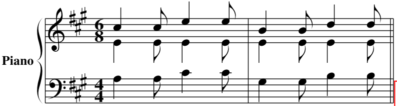
□

Sampling keeps what's essential

Sampling enables the music analyst to capture a view, or projection, of the target tune. For instance, given source



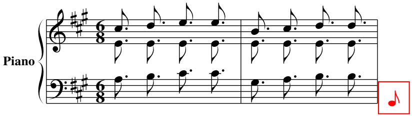
the following sample



removes rhythmic detail while keeping the main rhythmic structure, that given by rhythmic pattern , that is, $\frac{2}{8}, \frac{1}{8}$.

Sampling keeping the essential

Another sample, this time over $\frac{3}{16}$,



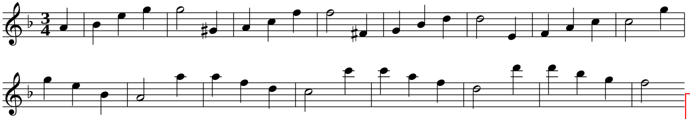
(which could be regarded as having meter $\frac{12}{16}$) keeps the melodic structure.

Epilogue

- When used together with the other combinators described in this series of slides, sampling offers support for musical analysis by **removing detail** (eg. passing notes, short rhythmic patterns) and providing a **view** (analysis) of the musical text.
- Melodic pattern identification calls for a **metric structure** in musical pitch enabling us to calculate the **derivative** of a melodic line, ie., the sequence of intervals involved.
- From melodic derivatives we can (re)build tunes again, by the converse operation of **integration**.
- Such will be the purpose of the next set of slides in this series.

Annex — an analysis of Schumann's opus 1

Theme of the Abegg Variations by Robert Schumann (1810-1856):



Source:

abegg_tune =
[("A", $\frac{1}{4}$), ("B", $\frac{1}{4}$), ("e", $\frac{1}{4}$), ("g", $\frac{1}{4}$), ("g", $\frac{1}{2}$), ("^G", $\frac{1}{4}$), ("A", $\frac{1}{4}$), ("c", $\frac{1}{4}$), ("f",

Annex — an analysis of Schumann's opus 1

Rhythmic analysis: getting the rhythm in the first place,

$$abegg\_r = map\ snd\ abegg\_tune =$$

$$[\tfrac{1}{4}, \tfrac{1}{4}, \tfrac{1}{4}, \tfrac{1}{4}, \tfrac{1}{2}, \tfrac{1}{4}, \tfrac{1}{4}, \tfrac{1}{4}, \tfrac{1}{4}, \tfrac{1}{2}, \tfrac{1}{4}, \tfrac{1}{4}, \tfrac{1}{4}, \tfrac{1}{4}, \tfrac{1}{2}, \tfrac{1}{4}, \tfrac{1}{4}, \tfrac{1}{4}, \tfrac{1}{4}, \tfrac{1}{2}, \tfrac{1}{4}, \tfrac{1}{4}, \tfrac{1}{4}, \tfrac{1}{4}, \tfrac{1}{2}, \tfrac{1}{4},$$

This appears to be a repetition of cell

$$cell1 = [\tfrac{1}{4}, \tfrac{1}{4}, \tfrac{1}{4}, \tfrac{1}{4}, \tfrac{1}{2}]$$

Let us check:

$$patternIndices\ cell1\ abegg\_r =$$
$$[(0,4),(5,9),(10,14),(15,19),(20,24),(25,29),(30,34),(35,39)]$$

Indeed: eight perfect copies of the cell (pretty classic!)

Annex — an analysis of Schumann's opus 1

As to the melody, we know the story already: word *Abegg* (surname of **Pauline von Abegg**, the young friend of the Schumann's) becomes

once paired with *cell1*. You obtain the above by running

```
abcPlay_ "F" "3/4" (zip "ABegg" cell1)
```

(Mind the need for `'B'` instead of `'b'` to obtain the right Abc pitch.)

Annex — an analysis of Schumann's opus 1

Let us now see how to obtain the retrograde inversion of the Abegg cell, which can be found in the start of the second part of the melody.

First, we define a generic function for retrograde inversion

$$retrog\ m = \mathbf{let}\ (l, r) = unzip\ m\ \mathbf{in}\ zip\ (reverse\ l)\ r$$

where *unzip* does what it says: splits a list of pairs in two lists.

Then we run

```
abcPlay_ "F" "3/4" (retrog (zip "ABegg" cell1))
```

and obtain

Annex — an analysis of Schumann's opus 1

The first part of the theme repeats the original cell four times, with different starting points,

while the second part does so for the inverted cell:

(Again pretty classic, for a romantic composer.)

Annex — an analysis of Schumann's opus 1

Having captured the architecture of the whole theme, what is left for us to study?

- The footholds of each repetition.

We can capture these by resorting to the *sample* function. Because of the anacrusis, the sampling pattern needs an extra crotchet (quarter note):

$$\mathit{bars} = [\tfrac{1}{4}] +\!\!+ \mathit{cycle}\ [\tfrac{6}{4}]$$

(an infinite sequence, as we want to sample as much as possible).

Let us do the sampling:

$$\begin{array}{l} \mathit{abcPlay\_}\ \texttt{"F"}\ \texttt{"3/4"}\ \mathit{stune} \\ \quad \textbf{where}\ \mathit{stune} = \mathit{sample}\ \mathit{bars}\ \mathit{abegg\_tune} \\ \qquad \mathit{bars} = [\tfrac{1}{4}] +\!\!+ \mathit{cycle}\ [\tfrac{6}{4}] \end{array}$$

Annex — an analysis of Schumann's opus 1

The outcome will be, for the first part:

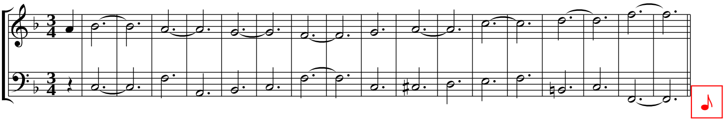
For the second part different footholds are required, as the cell is inverted. Altogether, the sampling rhythm will be

$$\mathit{bars} = [\tfrac{1}{4}] \mathbin{+\!\!+} \mathit{ntimes}\ [\tfrac{6}{4}]\ 3 \mathbin{+\!\!+} [\tfrac{9}{4}] \mathbin{+\!\!+} \mathit{cycle}\ [\tfrac{6}{4}]$$

and the overall outcome will be:

Annex — an analysis of Schumann's opus 1

Thus we reveal the "internal", chorale-like tune which underlies the whole theme, made of all footholds, together with the original bass (also suitably sampled):



The first part (corresponding to the ascending Abegg cell) is descending, the other is ascending.

A typical piece of German music, reminiscent of the Lutheran chorale style.

Annex — what makes music “jazzistic?”

A brief study of the transformations which lead from the Bourée of Bach’s Lute Suite in E Minor (BWV 996),

(here played by Narciso Yepes) to. . .

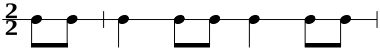
Annex — what makes music "jazzistic?"

. . . to Jethro Tull's piece with the same name (1969):

(From the LP "Stand up"; score available from the *Jethro Tull "Antology"*, ©1969 by Chrysalis Music Ltd., England)

What is a Bourée?

- Characteristic rhythmic pattern (bourée):

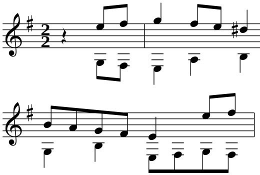


etc.

- Transformations could be melodic (eg. *blue notes*, etc) but in this example they will be all rhythmic. Let us see which.

The original: Bach's BWV 996 nr.5

Score sample:



Haskell script which generates the above:

$$(zip\ bwv996v1\ \ bwv996r1) \# (zip\ bwv996v2\ \ bwv996r2)$$

where number 1 refers to the top line and 2 to the bottom one, "v" means melodic voice, "r" means rhythmic pattern.

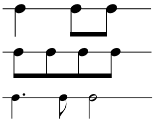
Analysis of Bach's BWV 996 nr.5

Definition of *bwv996r1*:

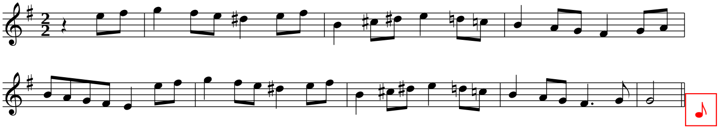
$$\mathit{bwv996r1} = \mathit{ntimes}\ \mathit{r11}\ 7 + \mathit{r12} + \mathit{ntimes}\ \mathit{r11}\ 6 + \mathit{r13}$$

where

$$\mathit{r11} = [\tfrac{1}{4}, \tfrac{1}{8}, \tfrac{1}{8}]$$
$$\mathit{r12} = [\tfrac{1}{8}, \tfrac{1}{8}, \tfrac{1}{8}, \tfrac{1}{8}]$$
$$\mathit{r13} = [\tfrac{3}{8}, \tfrac{1}{8}, \tfrac{1}{2}]$$



cf.



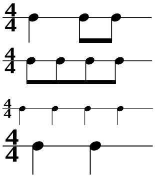
Analysis of Bach's BWV 996 nr.5

Definition of *bwv996r2*:

$$\mathit{bwv996r2} = \mathit{r11} \mathbin{+\!\!+} \mathit{ntimes}\ \mathit{r21}\ 3 \mathbin{+\!\!+} \mathit{r20} \mathbin{+\!\!+} \mathit{r12} \mathbin{+\!\!+} \mathit{ntimes}\ \mathit{r21}\ 3 \mathbin{+\!\!+} \mathit{minim}$$

where

$$\mathit{r11} = [\tfrac{1}{4}, \tfrac{1}{8}, \tfrac{1}{8}]$$
$$\mathit{r12} = [\tfrac{1}{8}, \tfrac{1}{8}, \tfrac{1}{8}, \tfrac{1}{8}]$$
$$\mathit{r21} = [\tfrac{1}{4}, \tfrac{1}{4}, \tfrac{1}{4}, \tfrac{1}{4}]$$
$$\mathit{r20} = [\tfrac{1}{4}, \tfrac{1}{4}]$$



cf.

Transformations

Compound effect obtained by

- **syncopation**
- **broken** rhythmic **cells**

Syncopation (Wikipedia):

> *(...) syncopation occurs when a temporary displacement of the regular metrical accent occurs, causing the emphasis to shift from a strong accent to a weak accent.*

This effect is obtained by cutting-off some duration, as specified in parameter d of function

$$\begin{aligned} &sync :: (Num\ a) \Rightarrow a \rightarrow [a] \rightarrow [a] \\ &sync\ d\ [\,] = [\,] \\ &sync\ d\ (h : t) = (h - d) : t \end{aligned}$$

Transformations

In the case of BWV996v, compare the original

$$(zip\ bwv996v1\ \ bwv996r1) \# (zip\ bwv996v2\ \ bwv996r2)$$

with

$$(zip\ bwv996v1\ \ bwv996r1') \# (zip\ bwv996v2\ \ bwv996r2)$$

where

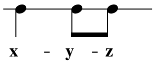
$$bwv996r1' = sync\ (\tfrac{1}{8})\ bwv996r1$$

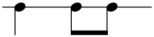
The top line is thus anticipated by a eigth-note, leading altogether to the following score.

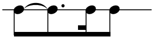
BWV996v after syncopation

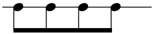
Breaking rhythmic cells

- Ian Anderson changes the cell $[x, y, z]$ characteristic of the bourée,



  into $[x + \frac{x}{4}, \frac{y}{2}, z]$.
- Thus cell $r11 = [\frac{1}{4}, \frac{1}{8}, \frac{1}{8}]$ 

  becomes $r11' = [\frac{5}{16}, \frac{1}{16}, \frac{1}{8}]$ 
- Consistently, cell

  $r12 = [\frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}]$ 

  becomes $r12' = [\frac{5}{32}, \frac{5}{32}, \frac{1}{16}, \frac{1}{8}]$ 

Breaking rhythmic cells

Broken-cells effect alone yields:

Breaking rhythmic cells

How do we do this? As follows:

$$\begin{aligned} &\mathit{dotted1}\ l = \mathit{apl}\ (\mathit{cycle}\ [(\tfrac{5}{4}*), (\tfrac{1}{2}*), \mathit{id}])\ l \\ &\mathit{dotted2}\ l = \mathit{apl}\ [(\tfrac{5}{4}*), (\tfrac{5}{4}*), (\tfrac{1}{2}*), \mathit{id}]\ l \end{aligned}$$

where *apl* nicely illustrates the power of functional programming:

$$\begin{aligned} &\mathit{apl} :: [a \rightarrow b] \rightarrow [a] \rightarrow [b] \\ &\mathit{apl}\ f\ l = \mathit{map}\ \mathit{ap}\ (\mathit{zip}\ f\ l) \end{aligned}$$

where *ap* applies functions to arguments:

$$\mathit{ap}\ (f, a) = f\ a$$

The two transformations together

Syncopated sentence of broken cells finally yields:

Last touch

A slightly different bass and some freedom in breaking the rhythmic cells will lead to Ian Anderson's version of Bach's BWV996v:

Musical text correlation

Mutual relationship between nr.65 of BWV 244 by J.S. Bach (*Mache Dich, Mein Herze, Rein air*),

...

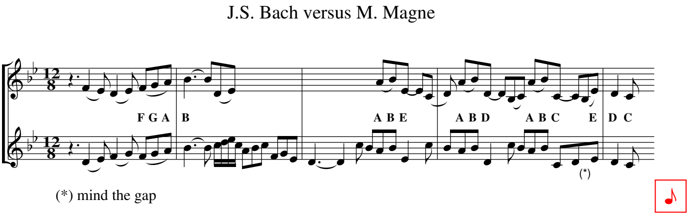
Musical text correlation

… and the theme of movie *Le Repos du guerrier* by Michel Magne (1930-1984)

— made popular by Frida Boccara (1940-1996) with the song *Cent mille Chansons*.

Musical text correlation

One can think of a program which produces several **correlations** between the two music sequences, in particular:



Is this sufficient for asserting that Magne's piece could have been inspired by such an air by Bach?