Computing for Musicology (0809.F104N5)
5. Towards Music Pattern Recognition

J.N. Oliveira

Dept. Informática,
Universidade do Minho
Braga, Portugal

June 2009
Licenciatura em Música
(http://www.musica.reitoria.uminho.pt/licenciatura.html)
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. (...)* **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:

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

  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 list are occupied with data satisfying a particular condition p:

$$\mathit{findIndices}\ p\ l = [i \mid (x, i) \leftarrow \mathit{zip}\ l\ [0\mathinner{.\,.}], p\ x]$$

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

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

and compare with

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

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 right component of pairs, yielding

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

Word (sequence) inversion

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

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

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

$$match\ p\ l\ i = p\ `isPrefixOf`\ (drop\ i\ l)$$

For instance, not only *isPrefixOf* `"Mendel"` `"Mendelssohn"` $=$ *True* holds, but also

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

Clearly,

$$isPrefixOf\ \ p\ l = match\ p\ l\ 0$$

Searching for patterns

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

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

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,

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

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 one that removes duplicates (*ndcopy*) by adding a filter at each stage:

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

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

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

  (all duplicates go out) with

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

  (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:

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

Removing locally repeated notes

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

Now the same once *nrep*'ed:

(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:

$$\mathit{tune} = [(\texttt{"c"}, 1\,\%\,8), (\texttt{"c"}, 1\,\%\,8), (\texttt{"c"}, 1\,\%\,8), (\texttt{"C"}, 3\,\%\,8), (\texttt{"e"}, 1\,\%\,8), (\texttt{"e"}, 1\,\%\,8), (\texttt{"e"}, 1\,\%\,8), (\texttt{"E"}, 3\,\%\,8), (\texttt{"g"}, 1\,\%\,8), (\texttt{"g"}, 1\,\%\,8), (\texttt{"g"}, 1\,\%\,8), (\texttt{"c"}, 1\,\%\,4), (\texttt{"e'"}, 1\,\%\,4), (\texttt{"c"}, 1\,\%\,4), (\texttt{"=B"}, 1\,\%\,4), \ldots]$$

Now the effect of *nrep*:

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

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:

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

Sampling for musical analysis

Two different samples of op.74iii,

and

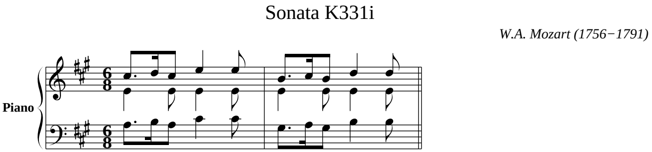
where the latter loses more information, keeping only the tonal thread.

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 is the purpose of the next set of slides in this series.