Introduction to Rhythmic C anons

Survey

Prelude ⇒
Algebrico molto: Representations & Algorithms ⇒
Andantino cantabile: canons from scratch ⇒
Stretta : canons from canons ⇒

Mc Master - july 11, 2005

1 Prelude

Init canonCrawler

```
<< Musica`musica`;
canonsCyclos[n_Integer] /; PrimeQ[n] := {n}
```

```
Off[BinaryFiles::"obslt"]
```

(un package devenu obsolète entre les versions 5.0 et 5.1, mais ça marche qd même)

```
canonsCyclos[n_Integer] :=
 Module[{divi, sb}, (* var locales *)
  divi = Drop[Divisors[n], 1];
  sb = Drop[Subsets[divi], 1] ;
  sb = Select[
      Map[ {#, Expand[Apply[Times , Map[cyclo , #]]]} &, sb],
       (* on obtient une liste d' e lem de la forme
         {un subset Z, ∏d ∈ z Φd  }     *)

    And[condT2[First[#]], rU01[Last[#]] ] &
   ];

  basicForm[polyToCanon@#, n] & /@ Last[Transpose@sb]
  (* on selectionne si les indices v e rifient
     T2 et si le polynomial est bien 0-1 *)
 ];

condT2[indCyclo_] := Module[
      {prodDesA, sA},
  sA = Select[indCyclo, primePowerQ];
  prodDesA = Flatten[Outer[Times, sA, sA]];
prodDesA = Select[prodDesA, ! primePowerQ[#] &];

  subsetQ[prodDesA, indCyclo]
 ]
```

```
fg[li_] := Module[{m},
  m = First[li]; While[MemberQ[li, m], m++]; m
 ]
(* finds first gap in a [sorted] list of Integers *)

(* motif est une liste tri e e d' entiers, commençant par 0 *)

pavePA[motif_] := Module[
  {soFar = motif, motifAugmente, firstGap, n = 0, canon = {motif},
   periodikQ = {}},

  intListe = Prepend[Drop[RotateLeft[motif] - motif, -1], 0];
  While[(firstGap = fg[soFar]) < Last[soFar],

   (* teste s' il reste des trous *)

   soFar = {0} ⋃ (Drop[soFar, firstGap] - firstGap);
   n = n + firstGap;

   (* inutile de garder la partie d e jà pleine *)

   motifAugmente = motif;
   While[motifAugmente ⋂ soFar ≠ {0},
    motifAugmente = motif + motifAugmente];
   soFar = soFar ⋃ (motifAugmente);
   AppendTo[canon, (n + motifAugmente)];
   If[MemberQ[periodikQ, soFar], Print["Infini periodique !"];
    Return[canon],
                    AppendTo[periodikQ, soFar] ]
      ];
  canon
 ]
```

```
goWild[tric_] :=
 Module[
  {soFar = tric,
   firstGap = 0,
   onset1 = {0},
   onset2 = {},
   cirt = Reverse[Max[tric] - tric]},
  If[Inner[LessEqual , cirt, tric, And],
          Reverse@goWild[cirt],
   (* l' algo marche SI le motif est "gauche" *)

   While[soFar ≠ Range[0, Last[soFar]],
    firstGap++;
    While[MemberQ[soFar, firstGap], firstGap++];
    Which[Not[MemberQ[soFar, firstGap + tric[[2]]]] &&
       Not[MemberQ[soFar, firstGap + tric[[3]]]],
         AppendTo[onset1, firstGap];
         soFar = soFar ⋃ (firstGap + tric),
       Not[MemberQ[soFar, firstGap + cirt[[2]]]] &&
       Not[MemberQ[soFar, firstGap + cirt[[3]]]],
         AppendTo[onset2, firstGap];
         soFar = soFar ⋃ (firstGap + cirt),
     True, Return[Print["problème !"], {onset1, onset2}]
             ]
   ];
   {onset1, onset2}
  ]]


plotWild[motif_] :=
 Module[{fitom, r1, r2},
  {r1, r2} = goWild[motif];
  fitom = (Max[motif] - motif // Reverse);
  plotCanon[Outer[Plus, r1, motif] ⋃
    Outer[Plus, r2, fitom]];
    ]
```

```
oPlus[a_, b_] := Outer[Plus,  b, a]
```

```
Off[General::"spell1"]
Options[selfSimMelody] = {
           tempo → 100, timeUnit → eighth,
           channel → 1, instrument → 5};


faireOrbites[n_, p_] :=
 Union[{{0}},
  Union /@ Table[Mod[p^i j, n], {j, n - 1}, {i, 0, n - 1}] ]

selfSimMelody[n_, p_, StHT_, opt___] :=
 Module[{orbits, nbOr, temp, chan , noteLength, pitches, faitNotes},
      orbits = Union[{{0}},
    Union /@ Table[Mod[p^i j, n], {j, n - 1}, {i, 0, n - 1}] ];
  ({ temp, chan, noteLength, instr} =
    { tempo, channel, timeUnit, instrument} /. {opt} /.
     Options[selfSimMelody]);

  nbOr = Length@orbits;
  pitches = StHT;
  While[Length[pitches] < nbOr,
       pitches = Join[pitches, StHT]
       ];
  pitches = Drop[pitches, nbOr - Length[pitches]];

  (* tout ça pour que pitches ait la même longueur que le
    nombre d'orbites *)

  faitNotes[pitch_, onsetsList_] := Module[{sol = {}},
    If[pitch ≠ 0, AppendTo[sol,
      {# × noteLength, Note[pitch, noteLength, temp, chan]} & /@
           onsetsList]];
       sol];

  Prepend[Partition[MapThread[
           faitNotes,
           {pitches, orbits}] // Flatten,
     2] // Sort,
   {0, ProgramChange[instr, chan]}
  ] ]
On[General::"spell1"]
```

```
Clear[repeter];
repeter[midiList_, n_] :=
 Module[{nbreEvts, timeUnit, i,  foo},
  foo = First[Transpose[midiList]];
  i = 1;
  While[(timeUnit = foo[[i + 1]] - foo[[i]]) == 0, i++];
  nbreEvts = 1 + Max[foo / timeUnit];
  Union @@
   Table[midiList /. {u_, v_} → {u + (k - 1) * nbreEvts * timeUnit, v},
    {k, n}]
      ]

Clear[dilater];
dilater[midiList_, n_, opts___] :=
     Module[
   {foo = midiList},
  foo /. {t_, Note[seq___]} → {n * t, Note[seq]}
      ]
Clear[changerInstrument];
changerInstrument[midiList_, instr_] :=
 (midiList /. ProgramChange[u_, chan_] → ProgramChange[instr, chan])

Clear[changerChannel];
changerChannel[midiList_, chan_] :=
 (midiList /. {ProgramChange[u_, ch_] → ProgramChange[u, chan],
    Note[a_, b_, c_, ch_] → Note[a, b, c, chan]})

Clear[changerDuree];
changerDuree[midiList_, dur_] :=
 (midiList /. Note[a_, b_, c_, d_] → Note[a,  dur, c, d])

Clear[transposer];
transposer[mid_, interval_] :=
     mid /. Note[a_, b___] → Note[a + interval, b]

Clear[decaler];
decaler[mid_, interval_] :=
 mid /. {a_, b_} → {a + interval, b}
```

```
ppmel[mel_] := Module[{n, sansZero,  yeti, height},
  n = Max[Flatten[sansZero = Drop[mel, 1]]];
  height = Length[sansZero];
  yeti[i_, j_] := MemberQ[sansZero[[i]], j] /. {True → 0, False → 1};
  ListDensityPlot[Table[yeti[i, j], {i, height}, {j, n}] ,
   AspectRatio → height / n, Ticks → {{0, Floor[n / 2], n}, {0, height}}]
 ]
```

```
Off[General::"spell1"];
Options[melodieToMidi] = {
       step → 1,
       timbre →  (ProgramChange[100, 1]),
       tempo → 100,
       basePitch → 55};

melodieToMidi[mel_, notes_, options___] := Module[

  {monTimbre, monStep, temp, bp, depart},
  ({monTimbre, monStep , temp, bp} =
    {timbre, step, tempo, basePitch} /. {options} /.
     Options[melodieToMidi]);
  depart = Min[Flatten[mel] ];

  Join[

   {{depart, TrackName["zikMu"]}},
   {{depart, TimeSignature[4, 4, 24, 8]}},
   {{depart, Tempo[temp * 1.]}},
   {{depart, KeySignature[0, 0]}},

   {{depart, monTimbre}},
   (* choix de l' instrument *)


   Map[{#[[2]] * eighth,
          Note[#[[1]], sixteenth, temp, 1]} &,

    Flatten[Map[
        Outer[List, {First[#]}, Drop[#, 1]] &,
                  MapThread[Prepend, {mel, notes}]],
      2]
   ],
    (* constitution de la melodie *)

   {{(2 + Max[Flatten@mel]) * eighth, EndOfTrack}}
       ]
 ]

On[General::"spell1"];
```

Some examples

■ Images

```
pp[{0, 2, 5, 7}, {0, 4, 8}];
```

```
pp[{0, 1, 4, 5}, {0, 2, 8, 10}];
```

```
plotCanon[{{0, 1, 4}, {2, 3, 6}, {5, 7, 13}, {8, 9, 12}, {10, 11, 14}}];
```

```
plotWild[{0, 2, 5}];
```

```
pp[{0, 1, 3, 4}, {0, 2, 3, 4, 5, 7}];
```

Observe the differences, of course there is a common philosophy: one same motif (sometimes undergoing a few transformations) enables an exact covering of all beats.

■ MIDI output

```
ExportMIDI[melodieToMidi[oPlus[{0, 2, 5, 7}, {0, 4, 8}],
  {55, 57, 59}, timbre → ProgramChange[95, 1]],
 "unCanonToutBête.mid"]
```

```
unCanonToutBête.mid
```

```
ExportMIDI[melodieToMidi[oPlus[{0, 1, 4, 5}, {0, 2, 8, 10}],
  {55, 57, 59, 56}, timbre → ProgramChange[95, 1]],
 "unCanonCompact.mid"]
```

```
unCanonCompact.mid
```

```
ExportMIDI[
 melodieToMidi[{{0, 1, 4}, {2, 3, 6}, {5, 7, 13}, {8, 9, 12},
   {10, 11, 14}}, {55, 57, 59, 64, 62}, timbre → ProgramChange[97, 1]],
 "unCanonJohnson.mid"]
```

```
unCanonJohnson.mid
```

```
goWild[{0, 2, 5}]
```

```
{{0, 1, 8}, {4, 11, 12}}
```

```
ExportMIDI[
 melodieToMidi[oPlus[{0, 2, 5}, {0, 1, 8}] ⋃ oPlus[{0, 3, 5}, {4, 11, 12}],
  {55, 56, 59, 60, 63, 64}, timbre → ProgramChange[109, 1]],
 "unCanonWild.mid"]
```

```
unCanonWild.mid
```

```
ExportMIDI[melodieToMidi[oPlus[{0, 1, 3, 4}, {0, 2, 3, 4, 5, 7}],
  8 + {58, 55, 61, 63, 67, 71}, timbre → ProgramChange[146, 1],
  tempo → 58],
 "unCanonMod2.mid"]
```

```
unCanonMod2.mid
```


A definition… among others

A rhythmic canon is a *disjoint* reunion of several *copies* (possibly transformed), of a *single* given motif; this union can be a range of integers, or a cyclic group like the usual canons cycling in time (ex. *Frère Jacques*).

For the most basic case, one only uses translates of a given motif A, meaning

$$A \bigcup (A + i_1) \bigcup (A + i_2) \bigcup \ldots (A + i_k) = \{0, 1, \ldots n - 1\}$$

either exactly, or after reduction modulo n. Let us put together all of the offsets $0, i_1, i_2 \ldots i_k$ in a set B;

then we may write

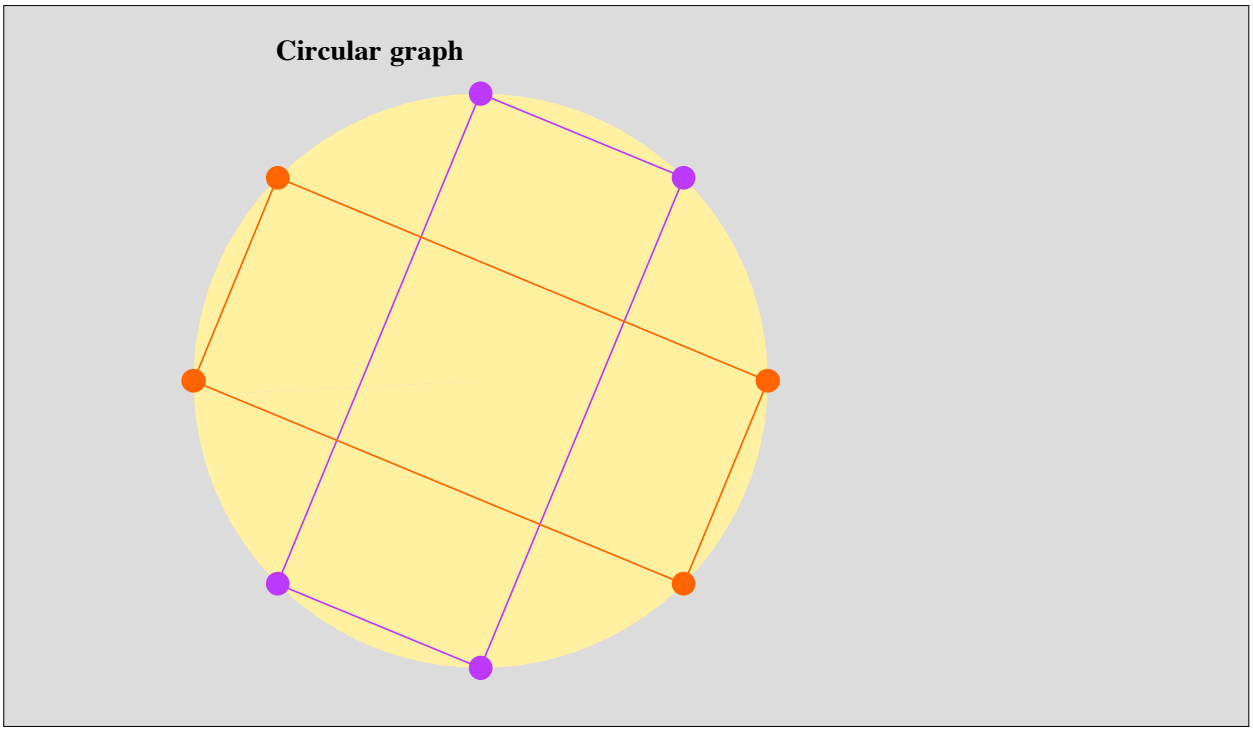
$$A \oplus B = \mathbb{Z}_n$$

The symbol $\oplus$ means that every number in $\mathbb{Z}_n$ is the result of one, and only one, addition.

In this most elementary (though far from trivial) case, A is usually called ***inner rhythm*** while B is the ***outer rhythm***.

Already the modelization takes us away from the musical surface: reduction modulo n means we envision a circular phenomenon, without beginning or end — quite unlike most musical scores ! Technically, this means a quotient structure, wherein we consider as equivalent several different objects: on the picture below, the motif is (0 1 4 5) and also (1 4 5 8) or even (2 3 6 7), etc…

Also noticeable on this picture is an interesting redundancy: a transposition (i.e. rotation of the circle) of a half-turn gives the same picture again (the motif has limited transpositions). It is strangely difficult to get a canon ***devoid*** of this L.T. phenomenon (a Vuza canon), as was proved by decades of work of worthwhile mathematicians !




```
clock[listeInt_, n_: 12, h_: 0.5] :=
    Graphics[{PointSize[0.04], Hue[h],
            Map[Point[{Sin[2 # π / n], Cos[2 # π / n]}] &, listeInt],
   contour[Map[{Sin[2 # π / n], Cos[2 # π / n]} &, listeInt] ]},
        AspectRatio -> Automatic]
contour[li_] := Line[Append[li, First[li]]]
```

```
Show[Graphics[{RGBColor[1, 0.934264, 0.570016], Disk[{0, 0}, 1] }],
      clock[{0, 1, 4, 5}, 8, 0.78], clock[{2, 3, 6, 7}, 8, 0.05],
      AspectRatio -> Automatic];
```

2 Algebrico molto: representations & algorithms

■ Interval sequences (basicForm)

As we could see on the last picture, the relevant information of a motif is invariant under rotations of the graph; that is to say under 'translation' (adding some fixed integer) + reduction modulo *n*.

In order to identify motifs up to these transformations, there is a very familiar way for North American music analysts: consider the sequence of all successive intervals; there are several different forms of this (up to circular permutations), one keeps the smallest one (i.e. with the biggest figures last).

For instance motif {0, 1, 4, 5} considered modulo 8 will be written (1, 3, 1, 3). This is similar (though not exactly identical) to Alan Forte's "basic form".

Computationally, one needs two functions, to navigate between the list of onsets and the interval list universes.

Also of interest are representations as characteristic functions (sequences of 0's and 1's). Sorting like above means choosing the ***Lyndon words***, which can be produced by fascinating and well documented algorithms. Very useful for enumeration of canons, but not my cup of tea.

■ Canon or not canon ?

If some sets *A*, *B* are given, one would like to know whether they tile a canon.

Notice that when the interval lists are given, we have inside the knowledge of the modulo to consider (the sum of all intervals); anyway one just needs to check whether

> $A \oplus B$ has exactly $|A| \times |B| = n$ (different) elements (mod. n)

Let us notice without any painful inquiry (yet), that determining whether a given set *A* will eventualy meet some *B* that can serve him as an *outer rhythm* is not an easy problem. It is indeed worse than 'not easy'; of course, one can try more or less all possible complements, which is an exponential time algorithm (some venture that the problem is NP), notice for instance that determination of *n* from the set *A* is far from obvious…

◂◂ ◂ ▸ ▸▸ 4 of 17

■ Pictures

Composers and mathematicians both appreciate good visual renderings. Pseudo-scores or clocks are the usual models (standard in software like *OpenMusic*™), some people unleash their creative power on this :

```
torusPpColor[{0, 1, 4, 7, 13, 24, 28, 37, 43, 48, 49, 52},
  {0, 8, 16, 18, 26, 34}];
```

■ Polynomials

From a set *A*, one easily gets a 0-1 polynomial (the converse is true):

```
A = {0, 1, 4, 5, 8, 9};
Plus @@ (X^A)
```

$$1 + X + X^4 + X^5 + X^8 + X^9$$

We thus enrich the universe; moreover, our sums-of-sets problems turn into factorisation questions:

```
Expand[(1 + X + X^4 + X^5 + X^8 + X^9) × (1 + X^2)]
oPlus[A, {0, 2}] // Flatten // Sort
```

$$1 + X + X^2 + X^3 + X^4 + X^5 + X^6 + X^7 + X^8 + X^9 + X^{10} + X^{11}$$

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}
```

Thus we can characterise canons with polynomials, for instance for 'canonical' canons:

$$A \oplus B = \mathbb{Z}_n \;\Leftrightarrow\; A(X) \times B(X) = 1 + X + X^2 + \ldots X^{n-1} \bmod X^n - 1 \;\; (\text{cond. } (T_0))$$

This is the starting point (the idea originates to ***Redeï*** around 1950, I think) of the mathematical study of rhythmic canons. As I was able to check in a number of different situations, the richer context of the algebra of polynomials enables the use of very powerful tools, and strong results on canons ensue.

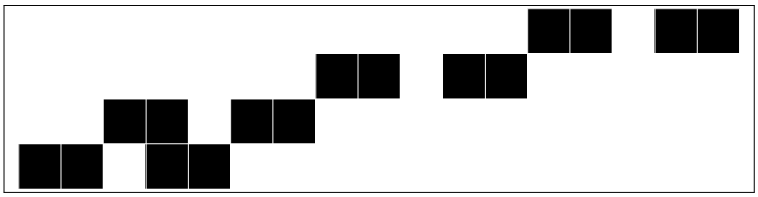
I will presently show something that *could not be done* without this polynomial stuff.

3 Andantino : canons from scratch

Are Simple Canons Really Simple ?

First things first: one cannot make a canon out of any motif. Not a ***canonical canon***, at any rate. For instance, the following motif won't ever tile a canon— you get either a rest or a chord !

```
pp[{0, 1, 3, 4}, {0, 2, 7, 12}];
```



It is conjectured that a technical condition, due to Ethan Coven and Aaron Meyerowitz (1998) (conditions $(T_1) + (T_2)$, it is Galois theory, cf. other conference…) would be necessary and sufficient for a given motif to ***tile***, e.g. to allow the

construction of a rhythmic canon. It was proved to be sufficient, at any rate, and this was enough to warrant implementation of some special canons in (the next version of) *OpenMusic™* (the lists below are interval lists):

The cyclo recipe

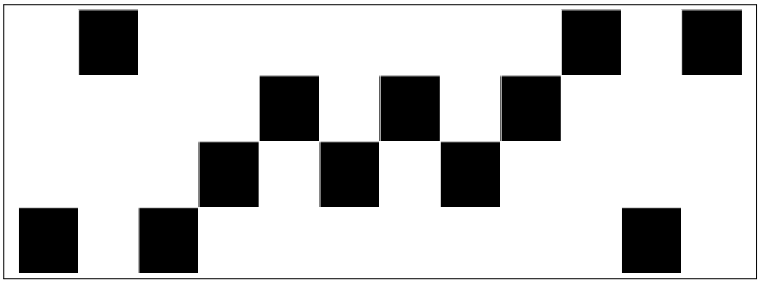
```
canonsCyclos[12]
```

```
{{1, 11}, {1, 1, 10}, {2, 10}, {1, 1, 1, 9}, {3, 9},
 {2, 2, 8}, {6, 6}, {1, 1, 1, 1, 1, 7}, {2, 1, 2, 7}, {1, 5, 1, 5},
 {1, 1, 4, 1, 1, 4}, {4, 4, 4}, {1, 3, 1, 3, 1, 3}, {3, 3, 3, 3},
 {2, 2, 2, 2, 2, 2}, {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}}
```

```
motif = canonsCyclos[12][[6]]
complete[motif]
ppBasic[motif, %[[1]], période → 12 ];
```

```
{2, 2, 8}
```

```
{{1, 5, 1, 5}, {3, 3, 3, 3}}
```



The recipe is a little obscure: take a given period n, then select some subsets of the divisor list of n (the criterion is about prime powers factors)

```
{condT2[{3, 4, 6, 12}], condT2[{2, 3, 12}]}
```

```
{True, False}
```

Then one computes some special polynomials Φ_i (the i^{th} cyclotomic polynomial) for i sweeping the selected list, and multiply all of them together. ***If*** the resulting polynomial is 0-1, it is guaranteed to provide a motif for some rhythmic canon — just shoot down the exponents of variable X.

```
cyclo /@ {3, 4, 6, 12}
Times @@ % // Expand
polyToCanon[%]
pp[%, {0, 3}];
```

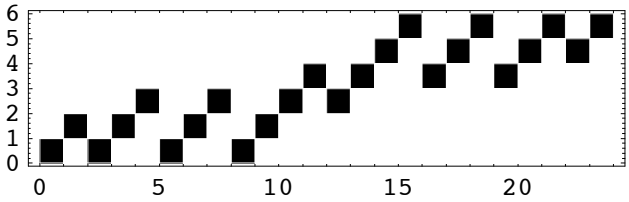
This is magical, and worse still, one cannot very well predict what will come out of this process (even the number of notes of the resulting motif is non obvious…) . Other techniques may be more rewarding in this respect.

Canons with retrogradation

■ Jonathan Wild's trichords theorem

It is a bit bleak to allow only translates of a given motif. Ol' Bach did it otherwise, let us emulate him and allow *retrogrades* of the motif:

```
plotWild[{0, 3, 6, 8}];
```



For a number of motives there is no such canon — try {0 1 3 4} again — but there is a wonderful result that Jon Wild proved for his PhD, namely that such a canon exists for ANY three note motif. Cherry on the whipped cream, the proof is actually an algorithm, indeed it is easy to build a solution by hand:

- Among the motif and its retrograde, chose the ***left*** form : say the motif is (0 3 5) , then consider rather (0 2 5).
- Drop a copy (translate) of this motif beginning with the first 'hole' on the right; repeat until you can't, or a canon has been obtained.
- If you can't do it with the left, do it with the right form.

Jon proved this ***always*** works. A huge mystery remains, though :

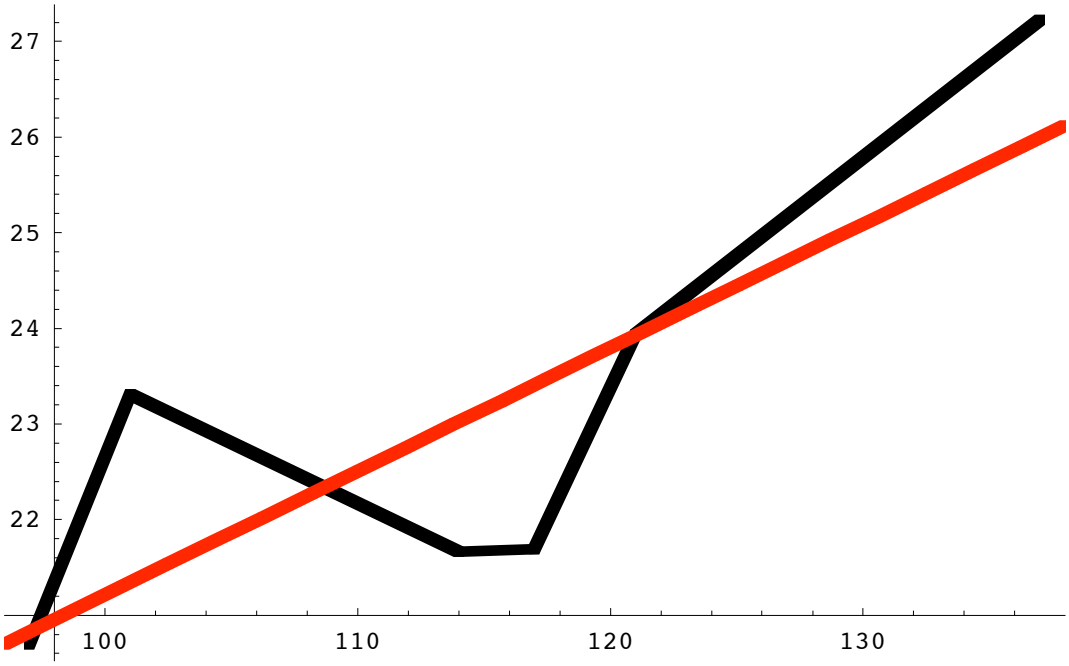
■ Questions about the 'lazy algorithm'

Conjecture : this algorithm always produces a palindrom

How strange, with a totally asymmetrical rule ! Of course, one wishes to try starting from right, using preferably the right form (thus you get the reverse solution, though I see no reason it should equate the direct one)… but it DOES NOT WORK if one starts from left with the right form !!! ex. (0 2 5).

Stranger still: there are (sometimes) other solutions, non palindromic…

Complexity: this algorithm may take a sizeable time to terminate, giving huge canons. Motif (0 3 7) already reaches 198 length, for (0 52 85) one gets 156 487 296 (!!!) (I did not try for worse…). The longest solutions seem exponential with respect to the width of initial motif. Here is a log-plot with regression line for the recordmen:



One last (open) question. As for 'simple' canons, there exist cases when this algorithm works, even for motifs longer than 3 notes (cf. (0 3 6 8)) — and of course some motifs won't comply ((0 1 3 4) again).

How would one characterise *a priori* motifs which tile with their retrograde ?

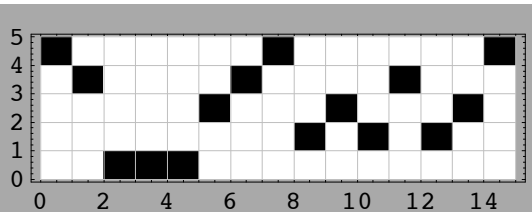
An aside there: there is a Wild canon with (0 3 6 8), which motif has been the basic rhythm of many tangos (since around 1915). It is possible to compose real music with this. I sincerely hope it sounds like an ordinary tango, your everyday-run-of-the-mill-friday-night-tango, (though memorable, superb specimen of *tango nuevo* style). I guarantee at least the composer always had in mind how it could be danced while writing it down.

Praise to laziness

■ Unturning the retrogradation

Wild's 'lazy algorithm' has a very general cardinal virtue: it runs !

I diverted the algorithm from Wild's original intention, allowing ***augmentations*** of the initial motif instead of ***retrogradation***. This is reminiscent of composer Tom Johnson's problem at the JIM in 2001, and also of his 'perfect square tilings', though with looser constraints.



Here is an algorithm :

⋆ pick a motif *A*

⋆ going rightwards, test among augmentations of *A* which one better (*=shorter) fills the remaining holes.

⋆ repeat until one of two cases:

- ✸ a canon is completed (tiling of a range, no more holes)
- ✸ we recognize a previously obtained situation (same pattern of holes).

■ What happens

Both cases occur.

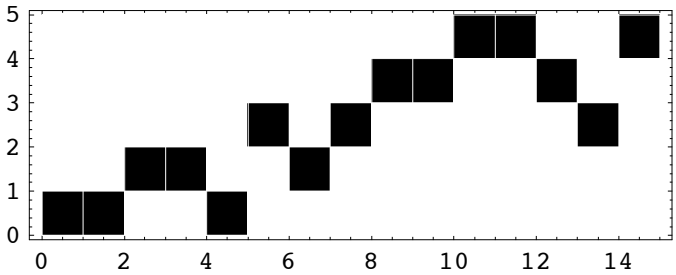
The former is a 'compact canon', i.e. tiling a range of integers.

The latter is a periodic canon, i.e. tiling of some $\mathbb{Z}_n$.

It is conceivable that the algorithm might go on and on without ever repeating itself. I have no ***proof*** this is possible (or impossible) though once I stopped arbitrarily the program after two days of computation…

First case: next example was initially (hand-)constructed by Tom Johnson in 2001 :

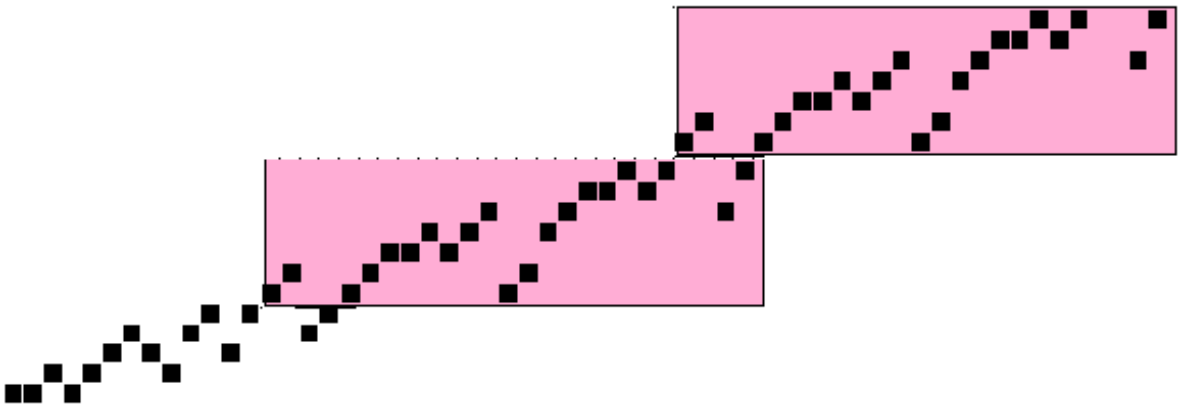
```
plotCanon[pavePA[{0, 1, 4}]];
```



Second case: with a much simpler motif (0 1 3) it is completely different :

```
ga = pavePA[{0, 1, 3}]
```

```
plotCanon[%];
```



9 of 17

■ Perspectives

Hence an open question (gaping, indeed):

is it ***always*** possible to make a rhythmic canon with ***any*** motif and (some of) its augmentations ?

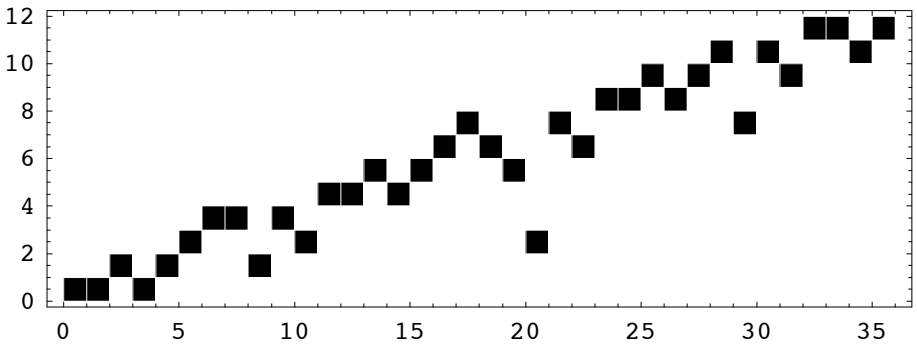
I feel the answer is yes, for cyclical canons anyway. But so little theoretically is known about these canons with augmentations that I won't even mention it. Perhaps we can even hope for tilings of a range which are not given by the lazy algorithm (for instance one exists for {0,1,3}).

One green light: one can AT LAST tile with the 'cursed' motif (0 1 3 4); it does make a pretty lengthy but catchy tune:

```
plotCanon[pavePA[{0, 1, 3, 4}], Frame → None];
```

Notice the 'lazy algorithm' is far from optimal (for instance the 'perfect square tilings' of Tom Johnson call for 'backtracking' programming). Let us turn back to (0 1 3) also: there are solutions tiling a range, i.e. without reduction modulo *n*:

```
plotCanon[{{0, 1, 3}, {0, 2, 6} + 2, {5, 10, 20}, {0, 1, 3} + 6,
   {0, 1, 3} + 11,
   {0, 2, 6} + 13, {0, 2, 6} + 16,
   {17, 21, 29}, {0, 1, 3} + 23, {0, 2, 6} + 25, {0, 2, 6} + 28,
   {0, 1, 3} + 32}];
```



10 of 17

Canons modulo *p*

This result is little known and fairly recent, first announced in Graz in mai 2004. The statement is very simple but the way to it was rather tough, I hope many pretty compositions will stem out of it.

In the spirit of this first conference, I'll drop the mathematics and focus on the music.

An other way to relax the constraint on canons is to allow, not ***exactly*** one note per beat, but a choice of numbers ("0 or 1" is a quite common choice that I won't study today). For largely mathematical reasons, to be honest, I studied the following case:

A canon modulo *p* is a rhythmic canon where on each beat the number of notes is *congruent* to 1 mod *p*.

In the easiest case: say $p = 2$, we allow any ***odd*** number of notes on each beat. The result is ***SO***simple (though not the proof thereof) :

> **For *ANY p* (prime), for *ANY* motif, there is a canon modulo *p*.**

I will sketch the proof in a later talk, is hangs on a Galois theorem about finite fields .

Anyway non-mathematicians can

try to find a solution by hand (beware ! some solutions are quite long), or…

use the ***canonCrawler.nb*** *Mathematica*™ library (online http://canonrythmiques.free.fr)

```
completeModulo[{0, 1, 3, 4}, 2]
```

```
{{0, 2, 3, 4, 5, 7}, 12}
```

```
pp[{0, 1, 3, 4}, First[%] , période → Last[%] ];
```

Like Wild's theorem, it is a kind of 'it always works' result.

4 Stretta : canons generating canons

■ Same period

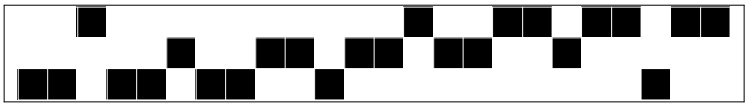
⋆ Back to square one, and 'simple' canons (more or less what follows can be adapted to most other kinds of canons).

Obviously, if a motif tiles a canon, so does its retrogradation, tiling the reverse canon.

More generally, if m is coprime with n the the map $x \to m\,x \bmod n$ is bijective (think of the cycle of fifths mod. 12). Hence from canon $A \oplus B = \mathbb{Z}_n$ one gets instantly $m\,A \oplus m\,B = \mathbb{Z}_n$.

■ Here is an example : *what did change in the transformation*?

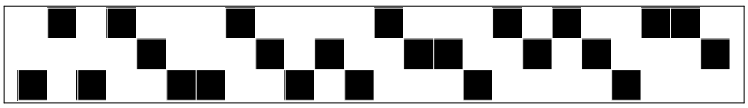
```
pp[{0, 1, 3, 4, 6, 7, 10, 21}, {0, 8, 16}, période → 24];
```



```
Sort /@ Mod[5 {{0, 1, 3, 4, 6, 7, 10, 21}, {0, 8, 16}}, 24]
```

```
{{0, 2, 5, 6, 9, 11, 15, 20}, {0, 8, 16}}
```

```
pp[{0, 2, 5, 6, 9, 11, 15, 20}, {0, 8, 16}, période → 24];
```



■ Midi output

```
ExportMIDI[
  melodieToMidi[Mod[oPlus[{0, 1, 3, 4, 6, 7, 10, 21}, {0, 8, 16}], 24],
   {55, 57, 59}, timbre → ProgramChange[95, 1]],
  "canonAffine1.mid"];
ExportMIDI[
  melodieToMidi[Mod[oPlus[{0, 2, 5, 6, 9, 11, 15, 20}, {0, 8, 16}], 24],
   {55, 57, 59}, timbre → ProgramChange[95, 1]],
  "canonAffine2.mid"];
ExportMIDI[
  melodieToMidi[Mod[oPlus[{0, 1, 8, 9, 16, 17}, {0, 2, 4, 22}], 24],
   {55, 57, 58, 60}, timbre → ProgramChange[95, 1]],
  "canonAffine3.mid"];
ExportMIDI[
  melodieToMidi[Mod[oPlus[{0, 5, 8, 13, 16, 21}, {0, 2, 4, 22}], 24],
   {55, 57, 58, 60}, timbre → ProgramChange[95, 1]],
  "canonAffine4.mid"];
```


■ Even more general, but not at all trivial

Thm (Vuza, 1990; Tijdeman, 1996)
Consider a canon $A \oplus B = \mathbb{Z}_n$
If m **is coprime with** n**, another canon is**
$m\,A \oplus B = \mathbb{Z}_n$.

We use this feature for the classification of canons. Having identified motifs up to a translation and now up to multiplication, we reach a classification of orbits under action of the affine group of $\mathbb{Z}_n$, a semi-direct product $\mathbb{Z}_n^* \triangleleft \mathbb{Z}_n$. Never mind that. In practice, it is already useful in order to ***reduce*** the sizes of lists of canons.

I will rub it in: one gets several different motifs A_1, A_2 … tiling with the ***same*** B. Musically speaking, the original motif suffers in the operation (not only suffers, but ciphers !).

```
Mod[5 {0, 1, 8, 9, 16, 17}, 24] // Sort
```

```
{0, 5, 8, 13, 16, 21}
```

```
pp[{0, 1, 8, 9, 16, 17}, {0, 2, 4, 22}, période → 24];
pp[{0, 5, 8, 13, 16, 21}, {0, 2, 4, 22}, période → 24];
```

⋆ It is convenient to make use also of ***duality:*** exchanging roles for A, B.

```
pp[{0, 2, 4, 22}, {0, 1, 8, 9, 16, 17}, période → 24];
pp[{0, 2, 4, 22}, {0, 5, 8, 13, 16, 21}, période → 24];
```

13 of 17

Other periods

We know several ways of turning a canon into a longer one.

■ Concatenation

Austrian researcher Harald Fripertinger made me realise the importance of this process. Very simple : take the motif written along the whole time-span of the canon, and repeat this motif a couple of times. The new motif again tiles a canon, furthermore it tiles with the same 'outer rhythm' (though a longer period):

If $A \oplus B = \mathbb{Z}_n$ then $(A + (A + n) + \ldots (A + (k-1)\, n)) \oplus B = \mathbb{Z}_{k\, n}$

This is trivial (as a proof we could sing "Frère Jacques" together for half an hour) but extremely interesting as a way of recursively generating all canons of given length (Fripertinger used this for generating and enumerating all canons up to $n = 119$).

```
pp[{0, 2, 3, 6, 9}, {0, 5}, période → 10];
pp[Outer[Plus, {0, 2, 3, 6, 9}, {0, 10}] // Flatten, {0, 5}, période → 20];
```

■ Midi output

```
ExportMIDI[melodieToMidi[Mod[oPlus[{0, 2, 3, 6, 9}, {0, 5}], 10],
   {55, 58}, timbre → ProgramChange[95, 1]],
  "concat1.mid"];
ExportMIDI[
  melodieToMidi[
   Mod[oPlus[Outer[Plus, {0, 2, 3, 6, 9}, {0, 10}] // Flatten, {0, 5}],
    20], {55, 58}, timbre → ProgramChange[95, 1]],
  "concat2.mid"];
ExportMIDI[melodieToMidi[Mod[oPlus[{0, 2, 7}, {0, 3, 6, 9}], 12],
   {55, 58, 60, 63}, timbre → ProgramChange[95, 1]],
  "stutt1.mid"];
ExportMIDI[
  melodieToMidi[Mod[oPlus[{0, 1, 4, 5, 14, 15}, {0, 6, 12, 18}], 24],
   {55, 58, 60, 63}, timbre → ProgramChange[95, 1]],
  "stutt2.mid"];
```

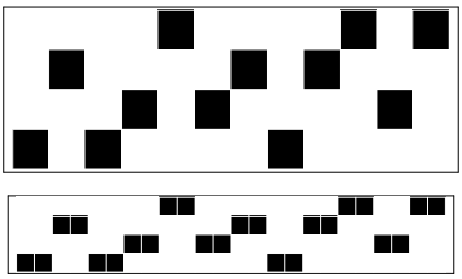
■ Stuttering

Replace each note (or rest) with k copies of itself. This time the 'outer rhythm' must be transformed too, by dilating the tempo with the same ratio k; it is of course also possible to exchange roles between both rhythms:

If $A \oplus B = \mathbb{Z}_n$ then $\left(A \otimes (1\,1\,\ldots\,1)_{k \text{ times}}\right) \oplus k\,B = \mathbb{Z}_{k\,n}$

The new motif may be seen as a reunion of several intertwined copies of the old one:

```
pp[{0, 2, 7}, {0, 3, 6, 9}, période → 12];
pp[{0, 1, 4, 5, 14, 15}, {0, 6, 12, 18}, période → 24];
```



Each stuttered voice "Frè-frè-frè / re-re-re / Ja-ja-ja / cques-cques-cques" could be sung not by one but three distinct people (not stuttering)! Which leads us to a ***new*** and fascinating transform:

14 of 17

Multiplexing of Canons:
A canon out of several (mostly) different canons

I found this in a recent paper by ***Kolountzakis & Matolcsi*** about commutative algebra and **Fuglede's** conjecture, in a very general context. It generalises a technique mentioned a few years ago in the now famous paper of Coven-Meyerowitz. It is also a generalisation of 'stuttering', and it currently used in your mobile phone and TV dish/cable to carry several signals in one bandwidth:

Let $A_1, \ldots A_k$ be several 'inner rhythms' (motifs) tiling a canon of period n with the *same* 'outer rhythm' B :

$$\forall\ i = 1 \ldots k \quad A_i \oplus B = \mathbb{Z}_n$$

Then $A = \bigcup (k\,A_i + i)$ tiles with outer rhythm $k\,B$.

In practice this is done by hand : dilate the tempo (ratio k) and intertwine the k motifs. It works well (indeed it ***always*** works!) when $A_1 \ldots A_k$ are (part of) an orbit under action of the affine group, as explained above. If all the A_i are equal, we get the stuttering again.

Thm [C-M, 2.5]:
Any canon wherein 'outer rhythm' B is divisible by some number p is the result of multiplexing p smaller canons.

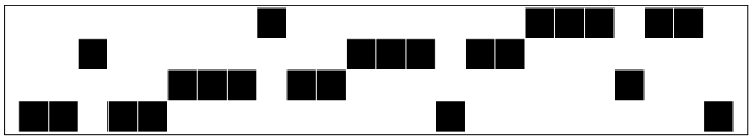
This theorem is not difficult. What is very interesting is that it applies to all VUZA canons hitherto constructed (Vuza canons are canons whose rhythms are not obtained by concatenation of smaller rhythms, they are quite rare)

Hence ***all known Vuza canons are reducible to simpler canons.***

But maybe we will discover other algorithms for Vuza canons in my next conference…

■ Example

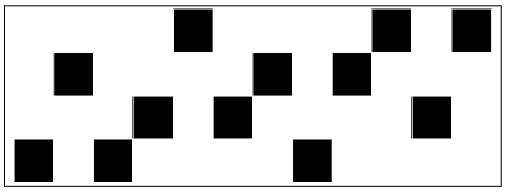
```
pp[2 * {0, 2, 7} ⋃ (1 + 2 * {0, 1, 11}), {0, 6, 12, 18},
  période → 24];
```

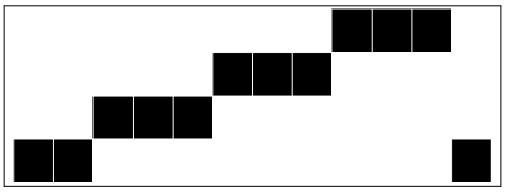


This canon has a six notes motif, it is multiplexed from the next two (three notes each) :

(look for the expanded version of the last one)

```
pp[{0, 2, 7}, {0, 3, 6, 9}, période → 12];
pp[{0, 1, 11}, {0, 3, 6, 9}, période → 12];
Show[GraphicsArray[{%%, %}]];
```





■ **MIDI output**

```
ExportMIDI[
  melodieToMidi[
   Mod[Join[oPlus[2 * {0, 2, 7} , {0, 6, 12, 18}] ,
     (oPlus[1 + 2 * {0, 1, 11}, {0, 6, 12, 18}])] , 24],
   {55, 57, 59, 60, 66, 68, 70, 72}, timbre → ProgramChange[95, 1]],
  "multiplexé.mid"];
ExportMIDI[melodieToMidi[Mod[oPlus[{0, 2, 7} , {0, 3, 6, 9}], 12],
   {55, 57, 59, 60}, timbre → ProgramChange[95, 1], tempo → 60],
  "multiplex1.mid"];
ExportMIDI[melodieToMidi[Mod[oPlus[{0, 1, 11} , {0, 3, 6, 9}], 12],
   {66, 68, 70, 72}, timbre → ProgramChange[95, 1], tempo → 60],
  "multiplex2.mid"];
ExportMIDI[
  melodieToMidi[
   Mod[Join[oPlus[2 * {0, 2, 7} , {0, 6, 12, 18}] ,
     (oPlus[1 + 2 * {0, 1, 11}, {0, 6, 12, 18}])] , 24],
   {55, 57, 59, 60, 55, 57, 59, 60}, timbre → ProgramChange[95, 1]],
  "multiplexFinal.mid"];
```

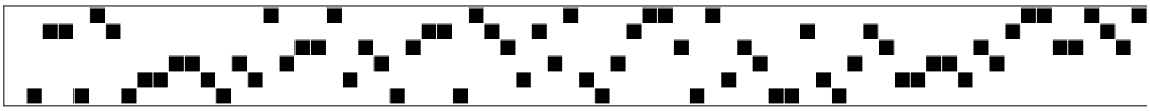
16 of 17

All Vuza 72 canons from multiplexing

(most lists here are interval lists)

```
{R72[[1]], S72[[1]]} // MatrixForm
ppBasic[R72[[1]], S72[[1]], période → 72];
```

```
({1, 3, 3, 6, 11, 4, 9, 6, 5, 1, 3, 20}
 {8, 8, 2, 8, 8, 38})
```



Procedure **eclate** takes a canon and demultiplexes it:

```
eclate[R72[[1]], S72[[1]], 2] // MatrixForm
```

```
({{3, 3, 12, 3, 3, 12}, {2, 10, 2, 10, 2, 10}}
 {4, 4, 1, 4, 4, 19})
```

Checking we have obtained (smaller) canons :

```
canonQ[{2, 10, 2, 10, 2, 10}, {4, 4, 1, 4, 4, 19}]
```

These are ***not*** Vuza canons (*A* & *B* acyclic) as seen from the score:

```
ppBasic[{2, 10, 2, 10, 2, 10}, {4, 4, 1, 4, 4, 19}, période → 36];
```

```
ppBasic[{3, 3, 12, 3, 3, 12}, {4, 4, 1, 4, 4, 19}, période → 36];
```

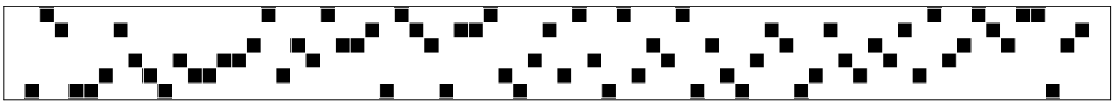
Stupefying : so Vuza 72 canons can be generated directly from **non** acyclic canons !!!

Procedure **youpla[{A_1, A_2 ...}, *B*]** does the multiplexing:

```
youpla[{{3, 3, 12, 3, 3, 12}, {2, 10, 2, 10, 2, 10}}, {4, 4, 1, 4, 4, 19}]
```

```
{{3, 3, 1, 5, 15, 4, 5, 6, 6, 3, 4, 17}, {8, 8, 2, 8, 8, 38}}
```

```
ppBasic[{3, 3, 1, 5, 15, 4, 5, 6, 6, 3, 4, 17}, {8, 8, 2, 8, 8, 38},
  période → 72];
```



Offsetting one component with varying values…

```
Table[Map[basicForm[#, 72] &,
   grosMélange[{{0, 3, 6, 18, 21, 24}, {1, 11, 13, 23, 25, 35} + k},
    {0, 4, 5, 9, 13, 32}]], {k, -1, 5}] // Union
```

```
{{{1, 3, 3, 6, 11, 4, 9, 6, 5, 1, 3, 20}, {8, 8, 2, 8, 8, 38}},
 {{1, 4, 1, 6, 13, 4, 7, 6, 6, 1, 4, 19}, {8, 8, 2, 8, 8, 38}},
 {{3, 1, 5, 6, 9, 4, 11, 6, 3, 3, 1, 20}, {8, 8, 2, 8, 8, 38}},
 {{3, 3, 1, 5, 15, 4, 5, 6, 6, 3, 4, 17}, {8, 8, 2, 8, 8, 38}},
 {{4, 1, 6, 6, 7, 4, 13, 6, 1, 4, 1, 19}, {8, 8, 2, 8, 8, 38}},
 {{4, 3, 6, 6, 5, 4, 15, 5, 1, 3, 3, 17}, {8, 8, 2, 8, 8, 38}}}
```

… one finds all the (inner rhythms of) acyclic canons usually called "R72" :

```
First[Transpose[%]]
```

```
{{1, 3, 3, 6, 11, 4, 9, 6, 5, 1, 3, 20}, {1, 4, 1, 6, 13, 4, 7, 6, 6, 1, 4, 19},
 {3, 1, 5, 6, 9, 4, 11, 6, 3, 3, 1, 20}, {3, 3, 1, 5, 15, 4, 5, 6, 6, 3, 4, 17},
 {4, 1, 6, 6, 7, 4, 13, 6, 1, 4, 1, 19}, {4, 3, 6, 6, 5, 4, 15, 5, 1, 3, 3, 17}}
```

■ **Code**

```
grosMélange[a_List, b_] :=
 Module[{k = Length[a]},
  {Flatten[Table[i - 1 + k a[[i]], {i, k}]], k b}]

youpla[a_List, b_] :=
 Module[{k = Length[a], n = Length[a[[1]] ] × Length[b]},
  n = n×k;
  {basicForm[Flatten[Table[i - 1 + k fromBasicForm@a[[i]], {i, k}]],
    n], basicForm[k fromBasicForm@b, n]}
 ]
(* the other way round *)

eclate[a_, b_, k_Integer] :=
 Module[{n = Length[a] Length[b], n0, aa = fromBasicForm@a},
  n0 = n / k;
  If[Union[Mod[fromBasicForm@b, k]] == {0},
   {Table[basicForm[ Select[aa, (Mod[#, k] == i) &] - i
                     ----------------------------------, n0],
                                  k
     {i, 0, k - 1}], b / k}]]

R72 = {{1, 3, 3, 6, 11, 4, 9, 6, 5, 1, 3, 20},
   {1, 4, 1, 6, 13, 4, 7, 6, 6, 1, 4, 19},
   {3, 1, 5, 6, 9, 4, 11, 6, 3, 3, 1, 20},
   {3, 3, 1, 5, 15, 4, 5, 6, 6, 3, 4, 17},
   {4, 1, 6, 6, 7, 4, 13, 6, 1, 4, 1, 19},
   {4, 3, 6, 6, 5, 4, 15, 5, 1, 3, 3, 17}};
S72 = {{8, 8, 2, 8, 8, 38}, {14, 8, 10, 8, 14, 18},
   {16, 2, 14, 2, 16, 22}};
```

17 of 17

Coda

This has been a general and informal journey over the land of rhythmic canons. Special emphasis was given to practical aspects and recipes for building canons, so that anyone can try his / her hand at them.

Of course specialised musical software (like *OM*) today contain extensive packages about canons: like musical scales or multiplication of chords, they are turning into one the basic tools for modern composers.