Speeding up Bewl

for applications to music theory

The link between music and topos theory

Thomas Noll, “The Topos of Triads” 2005

Start by considering the ***octave***, a set of 12 notes, and its distinguished subset, the chord of C major { C, E, G }

We consider transformations of the octave which preserve this chord.

They also need to preserve the ***affine structure*** of the octave. This means if two intervals have the same length, their images under the transformation do too.

There are precisely 8 such transformations - including the identity, which leaves every note of the octave fixed.

How do they look?

	1	a	b	c	f	F	g	G
1	1	a	b	c	f	F	g	G
a	a	a	a	a	a	a	a	a
b	b	b	b	b	b	b	b	b
c	c	c	c	c	c	c	c	c
f	f	b	c	b	F	f	b	b
F	F	c	b	c	f	F	c	c
g	g	c	a	a	a	a	G	g
G	G	a	c	c	c	c	g	G

The 8 transformations of the octave

These transformations form a ***monoid***: There's a unit element 1, and you can multiply any two elements to form a third.

It's called the ***triadic monoid*** *(T for short).*

By construction, we can multiply notes of the octave by elements of T. For example, B♭ times 1 is B♭ ; B♭ times a is C♮.

So the octave is a sort of vector space over T. (As you may remember, vectors can be multiplied by real numbers.) It's like working with vectors, only there is no addition! These '(multiplicative) actions of T' are called **musical objects**.

The octave, the chord of C major, and T itself are all musical objects.

You can add, multiply, and exponentiate musical objects. In fact...

Musical objects form a topos!

This means they can be manipulated via the four topos operations (+, *, ^, Ω) just as complex numbers, matrices, distributions, and even more abstract number-like objects can be manipulated with (+, -, *, /).

In fact this enables a whole rich arithmetic of musical objects, waiting to be explored, as Noll does in his paper. But the calculations are very awkward to do by hand. So we need the equivalent of a four-function calculator to do it – or better yet, a programmatic DSL.

Enter Bewl.

Constructing the triadic topos

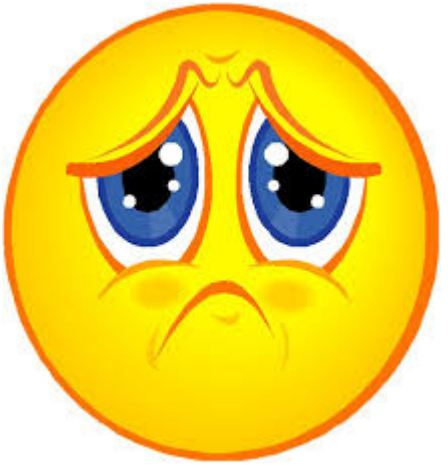
```
val (i, a, b, c, d, e, f, f2, g, g2) =
    ('i,'a,'b,'c,'d,'e,'f,'f2,'g,'g2)

val triadicMonoid =
    monoidFromTable(
        i, a, b, c, f,f2, g,g2,
        a, a, a, a, a, a, a, a,
        b, b, b, b, b, b, b, b,
        c, c, c, c, c, c, c, c,
        f, b, c, b, f2,f, b, b,
        f2,c, b, c, f,f2, c, c,
        g, c, a, a, a, a,g2, g,
        g2,a, c, c, c, c, g,g2
      )

val musicalObjects =
      FiniteSets.ToposOfMonoidActions of triadicMonoid
```

The musical objects are also an interesting example of a non-Boolean topos

Every topos has an ‘object of truth’ aka subobject classifier
For sets and permutations, this only has two values, ‘true’ and ‘false’
But in general, there are other truth-values, which have an arithmetic of their own; there are operations AND, OR, NOT, IMPLIES
The “law of excluded middle” **a v ¬a** fails. Also **¬¬a != a**
In the Bewl DSL, there is no **if-then-else**
You can still do maths in this crazy system: most of the usual rules hold. In fact the language is more expressive.
It’s a bit like non-Euclidean geometry, but with logic.
What can we say about music with this new language?

It’s All Too Slow

I can only just run the unit tests



But inspiration has struck, and there is now a new algorithm which will speed things up, a lot.

Against the odds, Bewl has been fairly performant so far. What's the hold up?

- **Code is too complex**

(Elements of products should be tuples
Good idea
Elements of exponentials should be functions
Terrible idea. Roll back
Although, thanks to Scala implicits, we can still use function notation)

- **Exponentials cause a combinatorial explosion**

Exponentiation is the third operation in the quartet (+, *, ^, Ω)

Given musical objects A and B, there is another one B^A representing the arrows from A → B , i.e. mappings that preserve T

Calculating B^A is monstrously inefficient

Exponentials

Suppose A, B are musical objects. To compute the exponential B^A, we have to enumerate all mappings A → B which respect the action of T
I was doing this by computing ALL maps A → B, then filtering, which is excruciatingly slow.

(Actually, it's even worse than that: the arrows from A to B are represented by the set of arrows from T x A to B, because they have a natural structure as a T-action. So in fact we have to calculate $B^{T \times A}$, which is unlikely to finish before the heat death of the observable universe.)

The solution is to find a set of ***generators*** of A. This is a subset of A which, when multiplied by T, gives all of A. Then by linearity, a mapping on A is completely determined by its action on the generators. (The generators are like the x-y-z coordinate system which lets us express any vector in 3-dimensional space in terms of only 3 vectors.)

Can then enumerate mappings via a simple backtracking algorithm!

The hard bit is to find the generators.

Finding generators

The widely used computational algebra package GAP doesn't have routines to do this! It covers rings, groups, modules, categories, semigroups but not monoid actions. (This is presumably because they are a degenerate structure despised by mathematicians.) So I had to plunge into the algebra:

Theorem:
Let A be a finitely generated action over a monoid.
Then the set of maximal monogenic subactions of A is finite, say $\{ M_i \}$;
their union is A.
Let G_i be the set of generators of M_i.
Then the minimal generating sets of A are precisely the transversals of the family $\{ G_i \}$. In particular, they all have the same size.

As an aside, this gives a theory of dimension for monoid actions. So they really are like vector spaces!

Better yet, it immediately yields an efficient one-pass *O(n log n)* algorithm for calculating a minimal set of generators.

The sketchy descriptions I've given here can in fact be refined into code that computes presentations (generators and relations) of monoid actions, then uses those to enumerate linear mappings and construct exponential objects.

The algorithms seem to work, and to be a lot faster, but I'm still wiring them up with the rest of Bewl, which is not entirely straightforward.

(In detail, it has to work with all the abstraction layers, so there will need to be a concept of a **MonoidAssistant** which has a default implementation for each topos, so that there can then be a special optimized version for the topos of finite sets (and perhaps for others) – like an OEM graphics driver.)

Once it's done though, I would expect to be able to verify Noll's calculations, and to extend them (24-tone music, anyone?) in Bewl.

THANK YOU

github.com/fdilke/bewl