Bewl

a programming language for topos theory

more precisely, a Scala DSL for the Mitchell-Benabou
internal language of a topos,
with some topos implementations

Summary

- About me
- Topos theory: a promising conceptual tool for escaping the limitations of set-based math
- It needs a "four function calculator":
this is why I wrote Bewl
- Engineering compromises: doing topos theory on a finite machine
- What can Bewl do so far?
- Future directions

About me, Felix Dilke

- I'm not an academic
- I'm a software developer on the team that maintains link.springer.com
- Bewl is my 10% time project at Springer Nature
- I've presented it to my colleagues, who are experienced real-world software developers with an interest in science
- So, this talk will be informal, even impressionistic, riddled with analogies between software and math, and I apologize in advance to domain experts if I seem to be making wild claims.
- Bewl is test-driven Scala code, open source on GitHub

The limitations of set-based math

A whole talk in itself, but briefly:

Colin McLarty's book on topoi has led me to see set theory
as a legacy platform like MS-DOS. Examples of anomalies:

- Permutation parity: an unexplained feature of the topos of sets (another motivation for Bewl was to explain parity).
- Large cardinals: advanced set theory perishing of its own contradictions, without useful application to mainstream math
- Ultrafilters: chimerical objects we can't construct

Topos theory can explain and perhaps alleviate these issues.

Topos theory

- A topos is a category with all the optional extras (finite products, equalizers, exponents, and subobject classifier)
- Inside a topos, one can define algebraic structures, quantifiers and a near-classical internal logic. So the topos becomes a workspace in which one can do math.
- A topos is an abstraction of the category of sets (the usual foundation for math) and shows how to do much of the same work in a much wider context.
- Driving metaphor: a topos is a "virtual machine, for math"
- Definitions, constructions, theorems "run" in a topos just as apps run on a VM, or SQL runs on a database

The promise

- Understand and escape limitations of set-based reasoning
- Cleanly separate language from implementation
  (just as webdesigners separate HTML from business logic)
- Example: Schur's lemma (that the endomorphisms of a simple module form a division ring) can be expressed
  in topos-valid form
- A lot of math can easily be "refactored" to apply
  in a much wider context

Examples of topoi

- Sets
- Smooth sets (a workspace for synthetic differential geometry)
- The effective topos (a workspace for computability)
- Sheaves (workspaces for algebraic geometry)
- Fuzzy sets (relative to a Heyting algebra)
- Diagrams of a given shape:

  graphs
  automorphisms
  monoid actions

Example: Schur's lemma

In an arbitrary topos, it's still true that endomorphisms of a simple module form a division ring

- Define a ring R as an object with arrows *: R x R -> R, unit: 1 -> R, subject to certain laws as arrow equalities
- Similarly an R-module has an abelian group structure and conditioned scalar multiplication *: R x M -> M
- "M is simple" is then expressed using a quantifier over M, saying that all submodules of M, i.e. subsets of M obeying certain closure laws, are equal to either 0 or M
- One can then formally construct the ring of endomorphisms as a subset of the exponential object M ^ M, and show that it obeys a law "for all x, either x = 0 or x has an inverse" in topos terms

Example: Schur's lemma (2)

The details of all this involve the Mitchell-Benabou internal language, which interprets logical formulas as statements about equality between arrows in the topos. A soundness result formalizes the proof as pure symbol-manipulation.

So now we have Schur's lemma for graphs, sheaves, monoid actions, etc.

Huge blocks of math translate similarly without much change. As a fancier example, one can do the same for the Los ultraproduct theorem (Volger 1975).

Already, many Bewl library methods are near-verbatim software transcriptions of proofs and definitions from math textbooks.

This all seems to point to:

An ambitious, Hilbertian programme of
"aggressively refactoring the foundations of math".

Obstacles:

- Topos logic is *intuitionistic*, i.e. allows
  multiple truth-values and no excluded middle
- Much classical math is irretrievably Boolean and
  can't easily be rewritten this way
  Example: number theory
- Basic concepts like finiteness and the real
  numbers don't have immediately clear analogues

There are potential answers to these, and they involve
fascinating conceptual questions.

- But most of all: How do you do the calculations?

Example: music theory

"The Topos of Triads", Thomas Noll, 2005

(paper on CiteSeer)

"The Topos of Music", Guerino Mazzola, 2002

(a book on SpringerLink)

Noll explores music theory through the topos of
actions over the "triadic monoid"

He had to do all these calculations by hand, for example
enumerating topologies on the triadic topos.

Bewl can now do some of these computations itself.
In particular, I verified that the C major chord
(modelled as an object in Noll's topos) is not injective.

Engineering challenges

To model topoi on a finite computer, I had to make some compromises.

- Although this isn't an iron rule, Bewl's topoi are locally finite, i.e. every $|\mathrm{Hom}(A, B)| < \infty$
- Digression: It turns out that this condition on a topos implies it has unique injective hulls. This result seems new. I wrote it up as a pure math paper on arXiv
- I also cache products and exponents. For objects A and B, Bewl lazily computes A x B and B ^ A just once.
- The word "object" is overloaded in computing, so in Bewl, there are "dots" and "arrows".

Engineering challenges (2)

- I use the 'cake' pattern to define a trait *Topos* as a stack of traits adding helper methods on top of *BaseTopos*
- For example, Bewl can calculate coproducts from the other topos operations. This is a verbatim transcript of constructions in McLarty/Moerdijk & Maclane from pure math into software.
- Every dot has a type attached to it. So a DOT[T] can be loosely thought of as ranging over values of type T. Functions of type T => U are easily interchangeable with arrows T > U (Scala sugar for the Bewl type >[T, U]).
- The main differences between functions and arrows are that arrows know their source and target, and can be compared for equality.

Engineering challenges (3)

- With its terse style, many expressive idioms, and advanced type system, Scala was an almost perfect fit for Bewl (I first tried writing it in Java, then Clojure)

For example, if dotA is a DOT[A] and dotB is a DOT[B], we can construct a new arrow like this:

```
val arrow: A > B =
    dotA(dotB) { a =>
        // ...
        <expression of type B>
    }
```

We can also apply arrows directly as if they were functions:

```
val a: A = ...
val b: B = arrow(a)
```

What are the values over which a dot ranges?

- A topos object (or "dot") in Bewl is a DOT[A], and calculations with it involve manipulating values of type A, as if the dot somehow ranged over values of type A.
- These values have meaning only inside the scope of an arrow definition, but it's all consistent with the very precise definition of the internal language as described in McLarty's book.
- An earlier version of the DSL interpreted the values a: A over which a DOT[A] "ranges" as arrows R > A, for some globally fixed "domain of definition" object R. Now they are pure syntax.

Tight integration

- Scala enables nice DSLs
- Some quite involved categorical calculations - for example, the tensorial strength axioms for strong monads
- can be described elegantly and concisely in Bewl.
- I've considered integrating the DSL with the language even deeper, using Scala implicit magic to make types and dots interchangeable.
- In summary, this all works a bit like the (mythical) """category Hask""", where objects and types are the same.

Truth values

- Since the topos may not be Boolean, Bewl has a type called TRUTH which essentially generalizes *Boolean*.
- The 'subobject classifier' in a topos is a DOT[TRUTH].
- Bewl autocalculates the logical operations on TRUTH values (and, or, implies, not) so you can do the equivalent of Boolean algebra.
- In fact, as the subobject classifier of a topos, DOT[TRUTH] is endowed with the structure of a Heyting algebra.

Which brings us to algebraic structures in Bewl.

Algebraic structures in Bewl

- The DSL lets you define these yourself, using off-the-peg operations (+, ~, etc) with known arities.

Example: Here's the definition of a commutative magma, a structure with one binary operation and one algebraic law:

```
    val commutativeMagmas =
        AlgebraicTheory(*)(α * β := β * α)
    case class CommutativeMagma[T](
        override val carrier: DOT[T],
        op: BinaryOp[T]
    ) extends commutativeMagmas.Algebra[T]
        (carrier)(* := op)
```

- The library includes definitions for monoids, groups, rings, actions, modules, lattices and Heyting algebras.

Algebraic structures (2)

- Actions and modules are slightly more involved - they define a new structure within the context of an existing one (monoids/groups and rings, respectively). The DSL caters for this via a concept of 'auxiliary scalars'.
- There are methods to calculate endomorphism monoids and automorphism groups, as algebraic structures in the topos.
- The library can easily be extended to add many familiar algebraic constructions (e.g. abelianizing a group) which translate easily in Bewl.

Topos implementations

Implementing topoi in Bewl is nontrivial.

There are four built-in implementations:

- Finite sets
- The topos of actions of a monoid
- The topos of actions of a group
- The topos of automorphisms

Topos implementations (2)

The last three all work in an existing topos.

So if Ɛ is a topos, we can construct a new topos Aut(Ɛ) consisting of all the dots-with-a-single-automorphism in Ɛ.

Similarly, if M is a monoid object in Ɛ, we can construct the topos of all objects A in Ɛ that come with an action of M, i.e. an arrow
A x M > A subject to the algebraic laws.

Exponentials, the subobject classifier, logical operations on truth values, etc will all be computed automatically.

Some example helper methods

Calculating the "name" of an arrow (see McLarty):

```
    trait Arrow[S <: ~, T <: ~] ... {
        ...
        final lazy val name: UNIT > (S → T) =
          (source > target).transpose(I) {
              (i, x) => arrow(x)
          }
    }
```

From the same class, tell if an arrow is epic:

```
    final lazy val isEpic: Boolean =
      target.exists(source) {
        (t, s) => target.=?=(
          t, arrow(s)
        )
      } toBool
```

Some example helper methods (2)

Similarly concise, generic library code uses the DSL to compute:

- is an arrow monic / epic / iso?
- is an object injective?
- 0, the initial object (coterminator)
- the arrow to any object from 0
- the inverse of an arrow, if it has one
- epi-mono factorizations

These work in any topos and for me, act as a major proof-of-concept validation for Bewl.

Performance

Better than you'd think, although there are no grounds for complacency.

I have largely managed to write code that is clean, efficient and DSL-compliant, thanks to the caching of common operations such as product and exponent.

Most of the required optimizations don't break any abstract layers and have been harmlessly packed away into 'driver extensions' for specific topoi.

Performance (2)

Remaining pain points:

- Quantifiers
- Computations with monoid actions

The latter were much improved by a special algorithm which efficiently calculates a presentation for a monoid action (!) so that Bewl can enumerate morphisms between two actions. This is obviously specific to sets.

The algorithm depends on a useful, but very elementary, criterion for finite generating sets of an monoid action.

Who might use this project?

Bewl definitely needs users. Some possibilities:

- People who want to use it as a learning aid to understand category theory. On the GitHub repo, I explain how to quickly set up a command-line REPL for this.
- Music theorists, continuing the Noll approach.
- Monoid and semigroup theorists, to explore the possibilities of topos-theoretic reasoning with actions
- anybody who understands both Scala and topoi, and wants to contribute or deepen their understanding!

Future directions

There is much more to do, but I'd especially like to add

- support for Lawvere-Tierney topologies and sheaves
- construct the topos of coalgebras for a left exact monad
- construct the double-exponential monad for an algebra

These are in principle fairly mechanical, and perhaps the hardest part is to construct decent test fixtures.

Volunteers welcome!

Future directions (2)

So many promising possibilities, so little time.

http://github.com/fdilke/bewl

THANK YOU