Partying with Permutations

Applying Bewl to the deep mystery of parity

Permutations

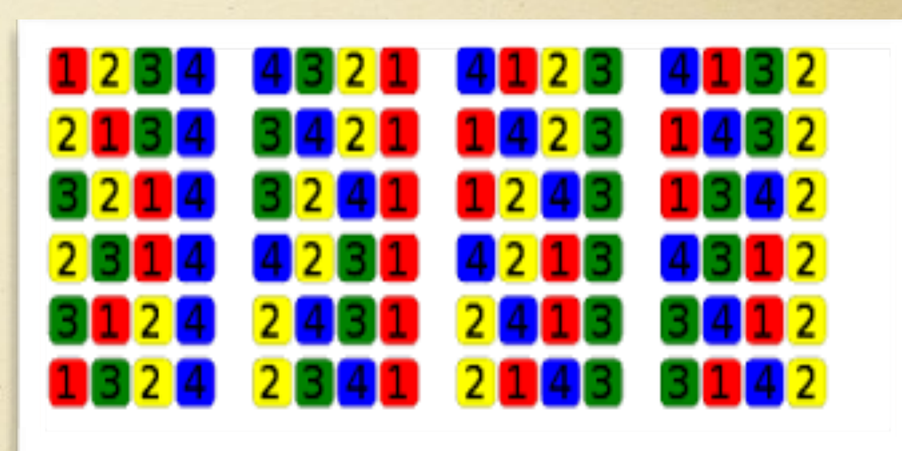
- A *permutation* is just a rearrangement of some finite set of objects.
- Example:
  (1, 2, 5)(3, 4)

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 5 & 4 & 3 & 1 \end{pmatrix};$$

- They come in two types: *even* and *odd*
- This respects multiplication:
  parity(A*B) = *parity*(A) * *parity*(B)
- Nobody knows why
- (You can define parity in terms of determinants, but that's no good because determinants are already defined in terms of parity)

Enter Bewl

- Bewl is a DSL which holds out the promise of new, unexplored languages for talking about toposes (systems of set-like objects)
- Permutations form a topos
- So, perfect fit…



Steps in applying Bewl

- Write a topos implementation for permutations
- Write a mini-DSL so that you can talk about permutations in an expressive way

```
val perm: Permutation[Int] = π(1,2)(3)π
```

- Get these in and out of the topos (HARD)

Getting things in and out of the topos is hard because…

- The topos presents a shiny, inscrutable abstraction layer
- There is a(n inadequate) wrapping mechanism for moving things and out, but…
- The permutations are actually a topos around a topos, because they are modelled as automorphisms of finite sets
- So you have to get through two layers of abstraction
- Types get tied in knots:

```
DOT[WRAPPER[Int] ⊕ WRAPPER[Int]]
```

Once that's done, here is my plan.

- For parity, the "flip" permutation Φ = (1,2) is obviously of central interest
- So let's crack open its structure
- The way to extract the structure of an object in a topos is to look at the theory of its double-exponential monad, Φ ^ Φ ^ _.
- So, I'll need to build the machinery for that…

…which will no doubt be an exciting voyage of discovery.

THANK YOU