Notes on Haskell

Tyler Foster

June 26, 2025

Contents

1 From category theory to Haskell

1.1 Types versus sets-with-added-structure

One major source of confusion when trying to understand Haskell from a category theoretical perspective is the role that *types* play in Haskell.

In many ways, category theory overrides the Bourbakian picture of mathematics, in which everything is build up from *sets with added structure* [Krö07] [Mar08]. Still, sets with added structure do form the most familiar examples of categories, and a large part of Haskell focusses on categories that arise from sets with added structure.

But Haskell discusses this notion of "sets with added structure" using the language of types, not the language of set theory. My rough heuristic for how to understand the two perspectives is:

> The *type* associated to a *set* is like the predicate that defines the set.
>
> The *set* associated to a *type* is the set of all instances of that type.

Think about how things work in an object-oriented scripting language like JavaScript or Python: the user defines new types by describing what internal structure and internal constraints any new instance of this type must have. It's a recipe for how new elements of some not-yet-fully-populated set will come into being.

Here's a little 2-column, informal dictionary to try to give some examples of what the distinction between types and sets-with-added-structure might be stressed from the perspective of familiar mathematical objects:

type	set with added structure
nonnegative integer less than n	$\{0, 1, 2, \ldots, n-1\}$
n-entry real coordinate vector	$\mathbb{R}^n$
prime ideal in $\mathbb{C}[x_1, x_2, \ldots, x_n]$	$\mathbb{A}^n_{\mathbb{C}}$
$m \times n$-matrix with entries in $\mathbb{F}_q$	$\mathrm{Mat}_{m,n}(\mathbb{F}_q)$
list of characters	$\{\text{all lists of characters}\} = \bigsqcup_{n=0}^{\infty} \mathrm{Char}^{\times n}$
program executable on present machine	$\{$ programs executable on present machine $\}$

This distinction might seem a bit odd from a purely mathematical perspective, but it makes more sense from a computer science perspective. In most computing environments, it is *much* easier to specify a condition approximating "n-entry real coordinate vector" than it is to instantiate the set $\mathbb{R}^n$.

[...]

1.1.1 What types and sets-with-added-structure reject from one another.

[...]

1. **Sets with axiomatic existence.**
2. **Sets with "hard-to-know" defining conditions.**
3. **Functions without defining formulae.**

[...]

1. **Distinct proofs of identity.**
2. **Recursive types.** [...nested lists...] [...bad Russell sets...]]
3. **Types that interact with physical/temporal reality.**

```
class Counter:
    count = 0

    def __init__(self):
        self.__class__.increment()

    @classmethod
    def increment(cls):
        cls.count += 1

    @classmethod
    def reset(cls):
        cls.count = 0
```

1.2 Comparing $\mathbf{Hask_{tot}}$ to Sets

1.3 The category $\mathbf{IO(Hask_{tot})}$.

[...]

1.3.1 Morphisms in IO($\mathbf{Hask_{tot}}$) as program-level combinators

1. Mapping the result (`fmap`, `>>=`)
2. Replacing the behavior (`const`, `ioB`)
3. Wrapping the behavior (`try`, `catch`, `log`)
4. Scheduling or deferring execution
5. Composing effects (`StateT`, `ReaderT`)

1.4 Other examples

1. `Maybe` Failure without error
2. `Either` Failure with error
3. `Reader` Read-only environment
4. `State` Stateful computation
5. `Writer` Logging
6. `Cont` Continuation-passing
7. `STM`

1.5 The Kleisli category $\mathbf{Kl_{IO}}$.

[...]

2 Haskell and optimization

[...]

2.1 Optimization between $\mathbf{Kl_{IO}}$ and $\mathbf{Hask_{tot}}$

References

[Krö07] Ralf Krömer. *Tool and Object: A History and Philosophy of Category Theory*, volume 32 of *Science Networks. Historical Studies*. Birkhäuser, Basel, 2007.

[Mar08] Jean-Pierre Marquis. *From a Geometrical Point of View: A Study of the History and Philosophy of Category Theory*, volume 14 of *Logic, Epistemology, and the Unity of Science*. Springer, Dordrecht, 2008.