Theories and Data Structures

—Draft—

ANONYMOUS AUTHOR(S)

ACM Reference format:
Anonymous Author(s). 2020. Theories and Data Structures
—Draft—. 1, POPL, Article 1 (January 2020), 4 pages.

DOI:

1 STARTED INTRODUCTION

It is relatively well-known in functional programming folklore that lists and monoids are somewhow related. With a little prodding, most functional programmers will recall (or reconstruct) that lists are, in fact, an instance of a monoid. But when asked if there is a deeper relation, fewer are able to conjure up "free monoid". Fewer still would be able formally prove this relation, in other words, to actually fill in all the parts that make up the adjunction between the forgetful functor from the category of monoids (and monoid homomorphisms) and the category of types (and functions) and the free monoid functor. To do so in full detail is, however, quite informative — and we will proceed to do so below. **Comment: MA** It is important to mention that this has been worked out in numerous other writings. That this is not the prime novelty of the work. E.g.; when a library claims to support X does it actually provide the necessaity 'kit' that that X /intersincly/ comes with? **End Comment**

So as to never be able to cheat, cut corners, etc, we will do all of our work in Agda, with this document[1] being literate (and, in fact, written in ultra-literate style via org-mode). But when we do, something interesting happens: we are forced to write some rather useful functions over lists. Somehow `map`, `_++_` and `fold` are all *required*.

But is this somehow a fluke? Of course not! So, what happens when we try to explore this relationship?

A programmer's instinct might be to start poking around various data-structures to try and see which also give rise to a similar relation. This is a rather difficult task: not all of them arise this way. Instead, we start from the opposite end: systematically write down "simple" theories, and look at what pops out of the requirements of having a "left adjoint to the forgetful functor". This turns out to be very fruitful, and the approach we will take here.

Naturally, we are far from the first to look at this. **Comment: JC** Fill in the related work here. From Universal Algebra through to many papers of Hinze **End Comment**. In other words, the *theory* behind what we'll be talking about here is well known.

So why bother? Because, in practice, there is just as much beauty in the details as there was in the theory! By *systematically* going through simple theories, we will create a dictionary between theories and a host of useful data-structures. Many of which do not in fact exist in the standard

[1] Sources available at https://github.com/JacquesCarette/TheoriesAndDataStructures

2020. XXXX-XX/2020/1-ART1 $15.00
DOI:

libraries of common (and uncommon) functional languages. And even when they do exist, all the "kit" that is derived from the theory is not uniformly provided.

Along the way, we meet several roadblocks, some of which are rather surprising, as results from the (theory) literature tell us that there really ought to be no problems there. Only when we dig deeper do we understand what is going on: classical mathematics is not constructive! So even when type theorists were busy translating results for use in functional programming, by not actually proving their results in a purely constructive meta-theory, they did not notice these roadblocks. **Comment: MA** Nice! **End Comment** Surmounting these problems will highlight how different axioms, via their *shape*, will naturally give rise to data-structures easily implementable with inductive types, and which require much more machinery.

In short, our contributions:

- a systematic exploration of the space of simple theories
- giving a complete dictionary
- highlighting the "kit" that arises from fully deriving all the adjunctions
- a survey of which languages' standard library offers what structures (and what kit)

2 TODO MONOIDS AND LISTS

Comment: JC Give the full details **End Comment**

```
module POPL19 where

open import Helpers.DataProperties

open import Function using (_∘_)
open import Data.Nat
open import Data.Fin  as Fin hiding (_+_)
open import Data.Vec as Vec hiding (map)
open import Relation.Binary.PropositionalEquality
```

3 TODO EXPLORING SIMPLE THEORIES

Comment: JC { **End Comment** Not fully sure how to go about this, while staying leisurely}

4 TODO TROUBLE IN PARADISE

Commutative Monoid, idempotence, and so on.

5 TODO SURVEY OF IMPLEMENTATIONS

6 TODO WE WANT TO BE SYSTEMATIC ABOUT

Exploring Magma-based theories: see https://en.wikipedia.org/wiki/Magma_(algebra) where we want to at least explore all the properties that are affine. These are interesting things said at https://en.wikipedia.org/wiki/Category_of_magmas which should be better understood.

Pointed theories: There is not much to be said here. Although I guess 'contractible' can be defined already here.

Pointed Magma theories: Interestingly, non-associative pointed Magma theories don't show up in the nice summary above. Of course, this is where Monoid belongs. But it is worth exploring all of the combinations too.

unary theories: wikipedia sure doesn't spend much time on these (see https://en.wikipedia.org/wiki/Algebraic_structure) but there are some interesting ones, because if the unary operation is 'f' things like forall x. f (f x) = x is **linear**, because x is used exactly once on each side. The non-linearity of 'f' doesn't count (else associativity wouldn't work either, as

* is used funnily there too). So "iter 17 f x = x" is a fine axiom here too. [iter is definable in the ground theory]

This is actually where things started, as 'involution' belongs here.
And is the first weird one.

Pointed unary theories: E.g., the natural numbers

Pointer binary theories: need to figure out which are expressible

more: semiring, near-ring, etc. Need a sampling. But quasigroup (with 3 operations!) would be neat to look at.

Also, I think we want to explore

- Free Theories
- Initial Objects
- Cofree Theories (when they exist)

Then the potential 'future work' is huge. But that can be left for later. We want to have all the above rock solid first.

7 TODO RELATIONSHIP WITH 700 MODULES

To make it a POPL paper, as well as related to your module work, it is also going to be worthwhile to notice and abstract the patterns. Such as generating induction principles and recursors.

A slow-paced introduction to reflection in Agda:
https://github.com/alhassy/gentle-intro-to-reflection

8 TODO TIMELINE

Regarding POPL:
https://popl20.sigplan.org/track/POPL-2020-Research-Papers#POPL-2020-Call-for-Papers
There is no explicit Pearl category, nor any mention of that style. Nevertheless, I think it's worth a shot, as I think by being systematic, we'll "grab" in a lot of things that are not usually considered part of one's basic toolkit.

However, to have a chance, the technical content of the paper should be done by June 17th, and the rest of the time should be spent on the presentation of the material. The bar is very high at POPL.

9 TODO TASK LIST ITEMS BELOW

- ☒ JC start learning about org mode
- ☒ JC Figure out how to expand collapsed entries
- ☐ JC See §4, first code block, of https://alhassy.github.io/init/ to setup `:ignore:` correctly on your machine.
  - This may require you to look at sections 2.1 and 2.2.

  This also shows you how to get 'minted' colouring.
- ☐ JC Write introduction/outline
- ☐ MA To read: *From monoids to near-semirings: the essence of MonadPlus and Alternative*, https://usuarios.fceia.unr.edu.ar/~mauro/pubs/FromMonoidstoNearsemirings.pdf.

10 DONE LITERATE AGDA IN ORG-MODE

JC, for now, use "haskell" labelled src blocks to get basic colouring, and I will demonstrate org-agda for you in person, if you like. Alternatively, I can generate coloured org-agda on my machine at the very end.

- A basic setup for *actually* doing Agda development within org-mode can be found at: https://alhassy.github.io/literate/
- Example uses of org-agda include
  - https://alhassy.github.io/next-700-module-systems-proposal/PackageFormer.html ; also · · · .org
    - Shallow use of org-agda merely for colouring ;; Prototype for Package Formers
      - Source mentions org-agda features that have not been pushed to the org-agda repo.
  - https://alhassy.github.io/PathCat/
    - Large development with categories ;; Graphs are to categories as lists are to monoids
  - https://github.com/alhassy/gentle-intro-to-reflection
    - Medium-sized development wherein Agda is actually coded within org-mode.