Theories and Data Structures

Jacques Carette, Musa Al-hassy, Wolfram Kahl

June 9, 2017

Abstract

$\boxed{\textbf{???}}$

This research is supported by the National Science and Engineering Research Council (NSERC), Canada

Contents

1 Introduction

We aim to show how common data-structures naturally arise from elementary mathematical theories.

In particular, we answer the following questions:

- Why do lists pop-up more frequently to the average programmer than, say, their duals: bags?
- More simply, why do unit and empty types occur so naturally? What about enumerations/sums and records/products?
- Why is it that dependent sums and products do not pop-up expicitly to the average programmer? They arise naturally all the time as tuples and as classes.
- How do we get the usual toolbox of functions and helpful combinators for a particular data type? Are they "built into" the type?
- Is it that the average programmer works in the category of classical Sets, with functions and propositional equality? Does this result in some "free constructions" not easily made computable since mathematicians usually work in the category of Setoids but tend to quotient to arrive in Sets? —where quotienting is not computably feasible, in Sets at-least; and why is that?

???

2 Overview

???

The Agda source code for this development is available on-line at the following URL:

`https://github.com/JacquesCarette/TheoriesAndDataStructures`

3 Forget

It is a common scenario where we have an algebraic structure with a single carrier set and we are interested in the categories of such structures along with functions preserving the structure.

We consider a type of "algebras" built upon the category of Sets —in that, every algebra has a carrier set and every homomorphism is a essentially a function between carrier sets where the composition of homomorphisms is essentially the composition of functions and the identity homomorphism is essentially the identity function.

Such algebras consistute a category from which we obtain a method to Forgetful functor builder for single-sorted algebras to Sets.

```
module Forget where

open import Level

open import Categories.Category using (Category)
open import Categories.Functor  using (Functor)
open import Categories.Agda     using (Sets)

open import Function2
open import Function
open import EqualityCombinators
```

[MA: *For one reason or another, the module head is not making the imports smaller.* **]**

A OneSortedAlg is essentially the details of a forgetful functor from some category to Sets,

```
record OneSortedAlg (ℓ : Level) : Set (suc (suc ℓ)) where
  field
    Alg         : Set (suc ℓ)
    Carrier     : Alg → Set ℓ
    Hom         : Alg → Alg → Set ℓ
    mor         : {A B : Alg} → Hom A B → (Carrier A → Carrier B)
    comp        : {A B C : Alg} → Hom B C → Hom A B → Hom A C
    .comp-is-∘  : {A B C : Alg} {g : Hom B C} {f : Hom A B} → mor (comp g f) ≐ mor g ∘ mor f
    Id          : {A : Alg} → Hom A A
    .Id-is-id   : {A : Alg} → mor (Id {A}) ≐ id
```

The aforementioned claim that algebras and their structure preserving morphisms form a category can be realised due to the coherency conditions we requested viz the morphism operation on homomorphisms is functorial.

```
open import Relation.Binary.SetoidReasoning
oneSortedCategory : (ℓ : Level) → OneSortedAlg ℓ → Category (suc ℓ) ℓ ℓ
oneSortedCategory ℓ A = record
  {Obj    = Alg
  ; _⇒_   = Hom
  ; _≡_   = λ F G → mor F ≐ mor G
  ; id    = Id
  ; _∘_   = comp
  ; assoc = λ {A B C D} {F} {G} {H} → begin⟨ ≐-setoid (Carrier A) (Carrier D) ⟩
    mor (comp (comp H G) F) ≈⟨ comp-is-∘                      ⟩
    mor (comp H G) ∘ mor F  ≈⟨ ∘-≐-cong₁ _ comp-is-∘           ⟩
    mor H ∘ mor G ∘ mor F   ≈˘⟨ ∘-≐-cong₂ (mor H) comp-is-∘   ⟩
    mor H ∘ mor (comp G F)  ≈˘⟨ comp-is-∘                     ⟩
    mor (comp H (comp G F)) ∎
  ; identityˡ = λ {{f = f} → comp-is-∘ ⟨≐≐⟩ Id-is-id ∘ mor f}
  ; identityʳ = λ {{f = f} → comp-is-∘ ⟨≐≐⟩ ≡.cong (mor f) ∘ Id-is-id}
  ; equiv     = record {IsEquivalence ≐-isEquivalence}
  ; ∘-resp-≡  = λ f≈h g≈k → comp-is-∘ ⟨≐≐⟩ ∘-resp-≐ f≈h g≈k ⟨≐≐⟩ ≐-sym comp-is-∘
  }
  where open OneSortedAlg A; open import Relation.Binary using (IsEquivalence)
```

The fact that the algebras are built on the category of sets is captured by the existence of a forgetful functor.

```
mkForgetful : (ℓ : Level) (A : OneSortedAlg ℓ) → Functor (oneSortedCategory ℓ A) (Sets ℓ)
mkForgetful ℓ A = record
  {F₀             = Carrier
  ; F₁            = mor
  ; identity      = Id-is-id $ᵢ
  ; homomorphism  = comp-is-∘ $ᵢ
  ; F-resp-≡      = _$ᵢ
  }
  where open OneSortedAlg A
```

That is, the constituents of a OneSortedAlgebra suffice to produce a category and a so-called presheaf as well.

Conclusion and Outlook

$\boxed{???}$