Category Theory
Monads in Agda

Yu-Yang Lin
1228863
MSci Computer Science

Individual Study 2 Presentation
School of Computer Science
University of Birmingham
21/12/2015

Exercise: The State Monad

Let S be a set (for some state, i.e. the type of the state/store).

We can define a monad on **Set** with $T\ A = S \to (S \times A)$

Let *State* be the monad:

- $State\ X = S \to (S \times X)$
- The Unit (η)

  $\eta : X \to (S \to (S \times X))$

  $\eta = x \mapsto (s \mapsto (s, x))$ where $s : S$ and $x : X$
- The Kleisli operator (equivalent to bind):

  Given $g : X \to (S \to (S \times Y))$

  $g^* : (S \to (S \times X)) \to (S \to (S \times Y))$

  $g* = f \mapsto s \mapsto (g(\pi_2\ f\ s))\ (\pi_1\ f\ s)$

Extension Task

Implement monads in Agda and verify the definition of the State monad is indeed a monad

Steps:

1. Implement categories
2. Implement the category **Set**
3. Implement monads
4. Implement the ***State*** monad on **Set**

Categories in Agda: Small Categories

To define a small category, we will need the following definition:

• Ob	: Set	Set of all objects
• Hom	: Ob → Ob → Set	Set of all morphisms
• id	: ∀ X : Ob .Hom X X	Identity morphism
• _;_	: ∀ X, Y, Z : Ob . Hom X Y → Hom Y Z → Hom X Z	Composition of morphisms

and the following laws:

- left-identity law
- right-identity law
- associativity law

Categories in Agda: Small Categories

This can be implemented in Agda:

```
record Cat : Set₁ where
  field Ob  : Set
        Hom : Ob → Ob → Set
        id  : {X : Ob} → Hom X X
        _▶_ : {X Y Z : Ob} → Hom X Y → Hom Y Z → Hom X Z

        left_id  : {X Y : Ob} {f : Hom X Y} → id ▶ f ≡ f
        right_id : {X Y : Ob} {f : Hom X Y} → f ▶ id ≡ f
        assoc    : {W X Y Z : Ob}
                   {f : Hom W X}
                   {g : Hom X Y}
                   {h : Hom Y Z} → f ▶ (g ▶ h) ≡ (f ▶ g) ▶ h
```

Example Small Category: Category of endofunctions on Booleans

First, we define the data types:

```
data Bool : Set where
  true  : Bool
  false : Bool

data Unit : Set where
  unit : Unit
```

Example Small Category: Category of endofunctions on Booleans

With this in place, we can define the category:

```
setB : Cat
setB = record
        { Ob   = Unit
        ; Hom = λ unit unit → Bool → Bool
        ; id  = λ b → b
        ; _▶_ = λ f g → λ x → g (f x)
        ; left_id  = refl
        ; right_id = refl
        ; assoc    = refl
        }
```

Categories in Agda: **Set**

With the previous definition, we were unable to define **Set**, however. To do this, we must define a universe level polymorphic record.

```
record Category {l : Level} : Set (lsuc l) where
  field Ob   : Set l
        Hom : Ob → Ob → Set l
        id  : {X : Ob} → Hom X X
        _▶_ : {X Y Z : Ob} → Hom X Y → Hom Y Z → Hom X Z

        left_id  : {X Y : Ob} {f : Hom X Y} → id ▶ f ≡ f
        right_id : {X Y : Ob} {f : Hom X Y} → f ▶ id ≡ f
        assoc    : {W X Y Z : Ob}
                   {f : Hom W X}
                   {g : Hom X Y}
                   {h : Hom Y Z} → f ▶ (g ▶ h) ≡ (f ▶ g) ▶ h
```

Categories in Agda: **Set**

With this definition, we can finally define **Set**:

```
set : Category
set = record
        { Ob   = Set
        ; Hom = λ X Y → Set → Set
        ; id   = λ X → X
        ; _▶_ = λ {X Y Z : Set} f g → λ x → g (f x)
        ; left_id   = refl
        ; right_id = refl
        ; assoc     = refl
        }
```

Monads in Agda

It is possible to define a monad in the traditional way:

- T — an endofunctor on C
- η — a unit natural transformation
- id μ — a multiplication natural transformation

The downside is that it would also require the definition of a functor and all the proofs associated. Alternatively, a monad can also be defined as a Kleisli Triple:

- T : Ob C $\rightarrow$ Ob C — a morphism in C
- η : X : Ob C $\rightarrow$ Hom_C(X,T X) — a unit transformation
- $(\_)^*$: X Y : Ob C $\rightarrow$ Hom_C(X,T Y) $\rightarrow$ Hom_C(T X, T Y) — the Kleisli operator

This is easier to implement since no definition of a functor is required, and no mention of naturality is given either. Hence I implemented monads as a Kleisli triple.

Monads in Agda

Given the following projections:

```
Ob      = Category.Ob
Hom     = Category.Hom
id      = Category.id
compose = Category._▶_
```

Monads in Agda

The definition of a Kleisli Triple is given as follows:

```
record Monad {l : Level}{C : Category {l}} : Set (lsuc l) where
  field T   : Ob C → Ob C
        η   : {X : Ob C} → Hom C X (T X)
        _⋆  : {X Y : Ob C} → Hom C X (T Y) → Hom C (T X) (T Y)

        monad_lid   : {X Y : Ob C}
                      {f : Hom C X (T Y)} →
                      compose C η (f ⋆) ≡ f
        monad_rid   : {X : Ob C} →
                      (η {X})⋆ ≡ id C {T X}
        monad_assoc : {X Y Z : Ob C}
                      {f : Hom C X (T Y)}
                      {g : Hom C Y (T Z)} →
                      compose C (f ⋆) (g ⋆) ≡ (compose C f (g ⋆)) ⋆
```

State Monad: Products

First, we need products defined:

```
data _×_ (A B : Set) : Set where
  _,_ : A → B → A × B

π₁ : {A B : Set} → (A × B) → A
π₁ (a , b) = a

π₂ : {A B : Set} → (A × B) → B
π₂ (a , b) = b

πid : {A B : Set}{p : A × B} → π₁ p , π₂ p ≡ p
πid {A}{B}{a , b} = refl
```

State Monad: Extensionality

Assuming function extensionality:

```
postulate exten : {X Y : Set}{f g : X → Y} →
                  ((x : X) → f x ≡ g x) → (f ≡ g)

exten2 : {X Y Z : Set}{f g : X → Y → Z} →
         ((x : X)(y : Y) → f x y ≡ g x y) → (f ≡ g)
exten2 h = exten (λ x → exten(h x))
```

State Monad: Definition

```
state : Set → Monad {lsuc lzero}{set}
state S = record
          { T  = λ A → S → (S × A)
          ; η  = λ a → λ s → (s , a)
          ; _⋆ = λ g f → λ s → g (π₂ (f s)) (π₁ (f s))
          ; monad_lid   = refl
          ; monad_rid   = lemma_rid
          ; monad_assoc = refl
          }
          where
            lemma_rid : {A : Ob set} →
                        (λf s → π₁ (f s) , π₂ (f s))
                        ≡ id set {S → (S × A)}
            lemma_rid {A} =
              begin
                (λ f s → π₁ (f s) , π₂ (f s))
                ≡⟨ exten2 (λ f s → πid {p = f s}) ⟩
                (λ f s → f s)
                ≡⟨ refl ⟩
                (λ f → f)
              ∎
```