# Монады и зависимость от контекста в семантике

О. Доманов

Новосибирск, 21 марта 2019 г.

# Теория

# Категория

- Набор объектов $A, B, ...$;
- Для каждых двух объектов $A, B$ существует набор морфизмов между ними $Hom(A, B)$. Обозначение: $f : A \to B, ...$ или $A \xrightarrow{f} B$.
- Для каждых двух морфизмов $f : A \to B$, $g : B \to C$ существует композиция $g \circ f : A \to C$;
- Композиция ассоциативна: $(f \circ g) \circ h = f \circ (g \circ h)$;
- Для каждого объекта существует единичный морфизм $id_A : A \to A$, такой, что $f \circ id_A = f = id_A \circ f$;

# Категория

- Набор объектов $A, B, \ldots$;
- Для каждых двух объектов $A, B$ существует набор морфизмов между ними $Hom(A, B)$. Обозначение: $f : A \to B, \ldots$ или $A \xrightarrow{f} B$.
- Для каждых двух морфизмов $f : A \to B$, $g : B \to C$ существует композиция $g \circ f : A \to C$;
- Композиция ассоциативна: $(f \circ g) \circ h = f \circ (g \circ h)$;
- Для каждого объекта существует единичный морфизм $id_A : A \to A$, такой, что $f \circ id_A = f = id_A \circ f$;

- Функтор $F : \mathcal{C} \to \mathcal{D}$ отображает объекты и морфизмы категории $\mathcal{C}$ в объекты и морфизмы категории $\mathcal{D}$ с «сохранением структуры»:
  - $F(id_A) = id_{F(A)}$
  - $F(f \circ g) = F(f) \circ F(g)$.

# Теория категорий и теория типов

Существует соответствие между теорией типов и теорией категорий:

$$\begin{array}{lcl} \text{Объекты} & \Leftrightarrow & \text{Типы (множества)} \\ \text{Морфизмы} & \Leftrightarrow & \text{Функции} \end{array}$$

Таким образом, $Hom(A, B)$ соответствует типу функций $A \to B$.

# Теория категорий и теория типов

Существует соответствие между теорией типов и теорией категорий:

| | | |
|---|---|---|
| Объекты | $\Leftrightarrow$ | Типы (множества) |
| Морфизмы | $\Leftrightarrow$ | Функции |

Таким образом, $Hom(A, B)$ соответствует типу функций $A \to B$.

Соответствие не взаимно однозначное: для каждой теории типов имеется категория, но не наоборот.

Для нас это не имеет значения, поскольку у нас основная семантика будет теоретико-типовой.

# Теоретико-типовая семантика

| | | |
|---|---|---|
| Нарицательные существительные | $\Rightarrow$ | Типы/объекты/множества |
| Термы | $\Rightarrow$ | Функции $t : A \to B$ |
| Пропозиции | $\Rightarrow$ | Типы/объекты/множества доказательств |
| Предикаты (проп. функции) | $\Rightarrow$ | Функции $A \to Set$ |
| Глаголы | $\Rightarrow$ | Функции $A \to Set$ |
| ... | $\Rightarrow$ | ... |

## Теоретико-типовая семантика

| | | |
|---|---|---|
| Нарицательные существительные | $\Rightarrow$ | Типы/объекты/множества |
| Термы | $\Rightarrow$ | Функции $t : A \to B$ |
| Пропозиции | $\Rightarrow$ | Типы/объекты/множества доказательств |
| Предикаты (проп. функции) | $\Rightarrow$ | Функции $A \to Set$ |
| Глаголы | $\Rightarrow$ | Функции $A \to Set$ |
| ... | $\Rightarrow$ | ... |

В случае интенсиональной семантики типы $A$ превращаются в типы $W \to A$.

Например, вместо Столица : Страна → Город мы имеем функции Столица : Страна → ($W$ → Город).

# Теоретико-типовая семантика

| | | |
|---|---|---|
| Нарицательные существительные | $\Rightarrow$ | Типы/объекты/множества |
| Термы | $\Rightarrow$ | Функции $t : A \to B$ |
| Пропозиции | $\Rightarrow$ | Типы/объекты/множества доказательств |
| Предикаты (проп. функции) | $\Rightarrow$ | Функции $A \to Set$ |
| Глаголы | $\Rightarrow$ | Функции $A \to Set$ |
| ... | $\Rightarrow$ | ... |

В случае интенсиональной семантики типы $A$ превращаются в типы $W \to A$.
Например, вместо Столица : Страна $\to$ Город мы имеем функции Столица : Страна $\to (W \to$ Город).

Мы хотим перенести (расширить) операции с экстенсионалами на операции с интенсионалами.

# Теоретико-типовая семантика

| | | |
|---|---|---|
| Нарицательные существительные | $\Rightarrow$ | Типы/объекты/множества |
| Термы | $\Rightarrow$ | Функции $t : A \to B$ |
| Пропозиции | $\Rightarrow$ | Типы/объекты/множества доказательств |
| Предикаты (проп. функции) | $\Rightarrow$ | Функции $A \to Set$ |
| Глаголы | $\Rightarrow$ | Функции $A \to Set$ |
| ... | $\Rightarrow$ | ... |

В случае интенсиональной семантики типы $A$ превращаются в типы $W \to A$.
Например, вместо Столица : Страна → Город мы имеем функции Столица : Страна → ($W$ → Город).

Мы хотим перенести (расширить) операции с экстенсионалами на операции с интенсионалами.

**Задача в общем виде:** мы имели типы $A, B, \dots$ с операциями на них, теперь мы имеем новые типы $MA, MB, \dots$ и хотим перенести наши операции на них.

# Понятие вычисления

- E. Moggi (1991) — монада как «понятие вычисления».
- $MA$ — вычисление со значениями в $A$, но с возможной дополнительной структурой.
- Помимо $MA = W \to A$:
  - $MA = \mathcal{P}A$ — индетерминизм;
  - $MA = A \times S$, где $S$ — множество состояний;
  - $MA = A + \text{Nothing}$ — результат, возможно, не определён.

# Категория Кляйсли (Kleisli)

Задача: расширить операции с $A$ на операции с $MA$.

- Те же объекты, но морфизмами считаем не $f : A \to B$, а $\hat{f} : A \to MB$.
- Нужно определить тождественную функцию и композицию функций для вычислений $\hat{f} : A \to MB$.
- Тождественная функция $\eta_A : A \to MA$.
- Вместо $A \xrightarrow{f} B \xrightarrow{g} C$
  нужно
  $A \xrightarrow{\hat{f}} MB \xrightarrow{\hat{g}} MMC \xrightarrow{?} MC$.
- Необходима функция $\mu_A : MMA \to MA$.

# Категория Кляйсли (Kleisli)

Задача: расширить операции с $A$ на операции с $MA$.

- Те же объекты, но морфизмами считаем не $f : A \to B$, а $\hat{f} : A \to MB$.
- Нужно определить тождественную функцию и композицию функций для вычислений $\hat{f} : A \to MB$.
- Тождественная функция $\eta_A : A \to MA$.
- Вместо $A \xrightarrow{f} B \xrightarrow{g} C$
  нужно
  $A \xrightarrow{\hat{f}} MB \xrightarrow{\hat{g}} MMC \xrightarrow{?} MC$.
- Необходима функция $\mu_A : MMA \to MA$.

Если $\eta, \mu$ (с нужными свойствами) существуют, то

1. функции Кляйсли $A \to MB$ составляют категорию;
2. мы имеем монаду.

# Функция связывания

Применение функции Кляйсли $f : A \to MB$ к $a : A$ описывается функцией

$$app : A \to (A \to MB) \to MB.$$

Мы будем использовать более удобную для нас функцию (bind)

$$\star : MA \to (A \to MB) \to MB.$$

# Функция связывания

Применение функции Кляйсли $f : A \to MB$ к $a : A$ описывается функцией

$$app : A \to (A \to MB) \to MB.$$

Мы будем использовать более удобную для нас функцию (bind)

$$\star : MA \to (A \to MB) \to MB.$$

Они связаны следующим образом:

$$\star(m, f) = \mu(app(m, Mf))$$

$$\star : MA \xrightarrow{app} MMB \xrightarrow{\mu} MB.$$

# Функция связывания

Применение функции Кляйсли $f : A \to MB$ к $a : A$ описывается функцией

$$app : A \to (A \to MB) \to MB.$$

Мы будем использовать более удобную для нас функцию (bind)

$$\star : MA \to (A \to MB) \to MB.$$

Они связаны следующим образом:

$$\star(m, f) = \mu(app(m, Mf))$$

$$\star : MA \xrightarrow{app} MMB \xrightarrow{\mu} MB.$$

Кроме того, будем писать $m \star f$, тогда можно строить цепочки

$$m \star f \star g \star \ldots$$

где $m : MA$, $f : A \to MB$, $g : B \to MC$, …

# Монада

- Эндофунктор в категории $\mathcal{C}$, то есть $M : \mathcal{C} \to \mathcal{C}$.
- Для любого $A \in \mathcal{C}$ существует $\eta : A \to MA$.
- Для любых $A, B \in \mathcal{C}$ существует $\star : MA \to (A \to MB) \to MB$.
- $m \star \eta = m$
- $\eta\ a \star f = f a$
- $(m \star f) \star g = m \star (\lambda x. f x \star g)$

# Монада

- Эндофунктор в категории $\mathcal{C}$, то есть $M : \mathcal{C} \to \mathcal{C}$.
- Для любого $A \in \mathcal{C}$ существует $\eta : A \to MA$.
- Для любых $A, B \in \mathcal{C}$ существует $\star : MA \to (A \to MB) \to MB$.
- $m \star \eta = m$
- $\eta\ a \star f = f a$
- $(m \star f) \star g = m \star (\lambda x. f x \star g)$

Reader или Function monad:

- $MA = W \to A$ — интенсионал
- $\eta\ a = \lambda w. a$ — rigid designator
- $m \star f = \lambda w. f(m(w))\ w$

# Формализация в Agda

# Синтаксис Agda

`B : Set`, `A : B` означает принадлежность типу.

# Синтаксис Agda

`B : Set`, `A : B` означает принадлежность типу.

Определение типа:

```
data C : Set where
  a b c : C
```

# Синтаксис Agda

`B : Set`, `A : B` означает принадлежность типу.

Определение типа:

```
data C : Set where
  a b c : C
```

Тип может быть зависимым:

```
data E : C → Set where
```

Тип $\Pi(x : C)\,Ex \quad \Rightarrow$ `g : ∀ (x : C) → E x`

Тип $\Sigma(x : C)\,Ex \quad \Rightarrow$ `h : Σ[ x ∈ C ] E x`

# Синтаксис Agda (2)

Определение функции:

```
f : C → D
f a = d
f b = e
f c = d
```

# Синтаксис Agda (2)

Определение функции:

```
f : C → D
f a = d
f b = e
f c = d
```

Аналогично записываются теоремы:

```
prf : Пропозиция
prf = d
```

# Синтаксис Agda (2)

Определение функции:

```
f : C → D
f a = d
f b = e
f c = d
```

Аналогично записываются теоремы:

```
prf : Пропозиция
prf = d
```

Или даже так:

```
_ : Пропозиция
_ = d
```

# Синтаксис Agda (2)

Определение функции:

```
f : C → D
f a = d
f b = e
f c = d
```

Аналогично записываются теоремы:

```
prf : Пропозиция
prf = d
```

Или даже так:

```
_ : Пропозиция
_ = d
```

Функции можно определять так:

```
_прибавить_ : C → D → D
```

Тогда вместо «`прибавить a b`» можно писать «`a прибавить b`».

```
data Мир : Set
  where w₁ w₂ : Мир
IntMonad = MonadReader Мир
```

Обычное обозначение для интенсионала: $^{\wedge}A$ (фактически, ∧ = M).

```
∧ : ∀ {i} → Set i → Set i
∧ = Reader Мир
```

```
mkIntensional : ∀ {a} {A : Set a} → (Мир → A) → ∧ A
mkIntensional f = mkReaderT (λ w → mkIdentity (f w))
```

Экстенсионал:

```
∨ : ∀ {i} {A : Set i} → ∧ A → (Мир → A)
∨ = runReader
```

⟦ m ⟧/ w — значение выражения (интенсионала) $m$ в мире $w$

```
⟦_⟧/_ : ∀ {i} {A : Set i} → ∧ A → Мир → A
⟦ ma ⟧/ w = ∨ ma w
```

rigid — функция $\eta$ (rigid designator):

```
rigid = return
```

```
data Человек : Set where
  John Mary Bill Unknown : Человек
```

## Пример: rigid designator John

```
iJohn : ∧ Человек
iJohn = rigid John

_ : ⟦ iJohn ⟧/ w₁ ≡ John
_ = refl

_ : ⟦ iJohn ⟧/ w₂ ≡ John
_ = refl

_ : ∀ (w : Мир) → ⟦ iJohn ⟧/ w ≡ John
_ = λ w → refl
```

## Более сложный пример.

```
Чемпион : ∧ Человек
Чемпион = mkIntensional f
  where
  f : Мир → Человек
  f w₁ = John
  f w₂ = Mary

_ : ⟦ Чемпион ⟧/ w₁ ≡ John
_ = refl

_ : ⟦ Чемпион ⟧/ w₂ ≡ Mary
_ = refl
```

Рассмотрим пример с зависимыми типами.

```
postulate
  _бежит-в-мире_ : Человек → Мир → Set
  Jr1 : John бежит-в-мире w₁
  Mr2 : Mary бежит-в-мире w₂
  Jr2⊥ : John бежит-в-мире w₂ → ⊥
  Br1 : Bill бежит-в-мире w₁

iБежит : Человек → (∧ Set)
iБежит h = mkIntensional (λ w → h бежит-в-мире w)


_ : ⟦ Чемпион ⋆ iБежит ⟧/ w₁ ≡ John бежит-в-мире w₁
_ = refl

_ : ⟦ Чемпион ⋆ iБежит ⟧/ w₁
_ = Jr1

_ : ⟦ Чемпион ⋆ iБежит ⟧/ w₂
_ = Mr2

_ : ⟦ rigid John ⋆ iБежит ⟧/ w₂ → ⊥
_ = Jr2⊥
```

То же для двуместного предиката.

```
postulate
  _любит_в-мире_ : Человек → Человек → Мир → Set
  MlB2 : Mary любит Bill в-мире w₂
  lself1 : ∀ (h : Человек) → h любит h в-мире w₁
  lself2 : ∀ (h : Человек) → h любит h в-мире w₂ → ⊥

iЛюбит : Человек → ∧ (Человек → Set)
iЛюбит h =
  mkIntensional (λ w → (λ h2 → h любит h2 в-мире w))

_ : ⟦ Чемпион ★ iЛюбит ⟧/ w₁ ≡ λ h → (John любит h в-мире w₁)
_ = refl

_ : ⟦ Чемпион ★ iЛюбит ⟧/ w₂ ≡ λ h → (Mary любит h в-мире w₂)
_ = refl
```

```
_ : (⟦ Чемпион ⋆ iЛюбит ⟧/ w₁) John ≡
                                John любит John в-мире w₁
_ = refl

_ : (⟦ Чемпион ⋆ iЛюбит ⟧/ w₁) John
_ = lself1 John

_ : (⟦ Чемпион ⋆ iЛюбит ⟧/ w₂) Mary → ⊥
_ = λ z → lself2 Mary z

_ : (⟦ Чемпион ⋆ iЛюбит ⟧/ w₂) Bill
_ = MlB2
```

Композиция вычислений.
«Отец чемпиона бежит».

```
iОтец : Человек → (∧ Человек)
iОтец h = mkIntensional (λ w → (отец h w))
  where
  отец : Человек → Мир → Человек
  отец John w₁ = Bill
  отец Mary w₁ = Bill
  отец Bill w₁ = Unknown
  отец Unknown w₁ = Unknown
  отец John w₂ = Bill
  отец Mary w₂ = John
  отец Bill w₂ = Unknown
  отец Unknown w₂ = Unknown
```

«Отец чемпиона бежит» = `Чемпион ★ iОтец ★ iБежит`:

```
_ : ⟦ Чемпион ★ iОтец ★ iБежит ⟧/ w₁ ≡ Bill бежит-в-мире w₁
_ = refl

_ : ⟦ Чемпион ★ iОтец ★ iБежит ⟧/ w₂ ≡ John бежит-в-мире w₂
_ = refl

_ : ⟦ Чемпион ★ iОтец ★ iБежит ⟧/ w₁
_ = Br1
```

Рассмотрим модальную логику.
Определяем отношение достижимости:

```
_≈>_ : Мир → Мир → Set
W₁ ≈> W₂ = ⊤
```

Миры, достижимые из некоторого мира.
Здесь `Σ[ w ∈ Мир ] P` обозначает $\Sigma(w : \text{Мир})P$.

```
acc : Мир → Set
acc w₀ = Σ[ w ∈ Мир ] w₀ ≈> w
```

Необходимость и возможность.

```
□ : ∧ Set → Set
□ P = ∀ w → ⟦ P ⟧/ w
◇ : ∧ Set → Set
◇ P = Σ[ w ∈ Мир ] ⟦ P ⟧/ w

□' : Мир → ∧ Set → Set
□' w₀ P = ∀ (x : acc w₀) → ⟦ P ⟧/ (proj₁ x)
◇' : Мир → ∧ Set → Set
◇' w₀ P = Σ[ x ∈ acc w₀ ] ⟦ P ⟧/ (proj₁ x)
```

«Необходимо, что чемпион бежит».

```
_ : □ (Чемпион ⋆ iБежит)
_ = prf where
  prf : (w : Мир) → ⟦ Чемпион ⋆ iБежит ⟧/ w
  prf w₁ = Jr1
  prf w₂ = Mr2

_ : □′ w₁ (Чемпион ⋆ iБежит)
_ = prf where
  prf : (w : acc w₁) → ⟦ Чемпион ⋆ iБежит ⟧/ (proj₁ w)
  prf (w₁ , _) = Jr1
  prf (w₂ , _) = Mr2
```

«Возможно, что чемпион бежит».

```
_ : ◇ (Чемпион ⋆ iБежит)
_ = w₁ , Jr1

_ : ◇′ w₁ (Чемпион ⋆ iБежит)
_ = (w₂ , _) , Mr2
```

## Сообщения о мнении. Ральф.

```
data НосительМнения : Set where
  Ральф Мы : НосительМнения
IntMonad = MonadReader НосительМнения

∧ : ∀ {i} → Set i → Set i
∧ = Reader НосительМнения

mkIntensional : ∀ {i} {A : Set i} → (НосительМнения → A) → ∧ A
mkIntensional f = mkReaderT (λ w → mkIdentity (f w))

∨ : ∀ {i} {A : Set i} → ∧ A → (НосительМнения → A)
∨ = runReader

⟦_⟧/_ : ∀ {i} {A : Set i} → ∧ A → НосительМнения → A
⟦ ma ⟧/ b = ∨ ma b

rigid = return
```

```
data Человек : Set where
  чел-в-шляпе чел-на-пляже Орткут : Человек

xh : ∧ Человек
xh = mkIntensional f
  where
  f : НосительМнения → Человек
  f Ральф = чел-в-шляпе
  f Мы = Орткут

xb : ∧ Человек
xb = mkIntensional f
  where
  f : НосительМнения → Человек
  f Ральф = чел-на-пляже
  f Мы = Орткут
```

```
postulate
  _считает-что_шпион : НосительМнения → Человек → Set
  spyh : Ральф считает-что чел-в-шляпе шпион
  spyb : Ральф считает-что чел-на-пляже шпион → ⊥

iСчитает-шпионом : Человек → (∧ Set)
iСчитает-шпионом h =
  mkIntensional (λ b → b считает-что h шпион)

_ : ⟦ xh ⋆ iСчитает-шпионом ⟧/ Ральф ≡
             Ральф считает-что чел-в-шляпе шпион
_ = refl

_ : ⟦ xh ⋆ iСчитает-шпионом ⟧/ Ральф
_ = spyh

_ : ⟦ xb ⋆ iСчитает-шпионом ⟧/ Ральф → ⊥
_ = spyb
```

```
_ : ⟦ rigid Орткут ⋆ iСчитает-шпионом ⟧/ Мы ≡
                        Мы считаем-что Орткут шпион
_ = refl

_ : ⟦ rigid Орткут ⋆ iСчитает-шпионом ⟧/ Ральф ≡
                        Ральф считает-что Орткут шпион
_ = refl
```

Невозможно доказать, что

```
_ : ⟦ rigid Орткут ⋆ iСчитает-шпионом ⟧/ Ральф
```

Спасибо за внимание!