Аппликативные функторы

Bartosz Milewski

Перевод:
Геннадий Чернышев
(https://henrychern.wordpress.com/)

В отличие от монад, которые пришли в программирование из теории категорий, аппликативные функторы берут свое начало в программировании. McBride и Paterson представили аппликативные функторы в качестве жемчужины программирования в своей статье «Аппликативное программирование с эффектами»[1]. Они также привели категорную интерпретацию аппликативов в терминах *строгих слабых моноидальных функторов*. Принято считать, что «монада — это моноид в категории эндофункторов», так что «аппликатив — строгий слабый моноидальный функтор».

Так называемая «тензорная прочность», по-видимому, важна в категорной семантике, и в своей основополагающей работе «Понятия вычислений и монад»[2] Moggi утверждал, что эффекты должны описываться с использованием строгих монад. Это имеет смысл, учитывая, что вычисления выполняются в контексте, и необходимо иметь возможность сделать глобальный контекст доступным в рамках монады. Тот факт, что мало говорится о строгих монадах в Haskell, объясняется тем, что все функторы в категории **Set**, которая лежит в основе системы типов Haskell, обладают канонической прочностью. Так почему же мы говорим о прочности, когда имеем дело с аппликативными функторами? Я изучил этот вопрос и пришел к выводу, что фундаментальной причины нет, и что можно просто сказать:

[1] http://www.staff.city.ac.uk/~ross/papers/Applicative.pdf
[2] https://core.ac.uk/download/pdf/21173011.pdf

Здесь я расскажу о различных эквивалентных категорных определениях аппликативного функтора. Я начну со слабого замкнутого функтора, затем перейду к слабому моноидальному функтору и покажу эквивалентность двух определений. Затем я представлю исчисление концов и покажу, что третье определение аппликативного функтора как моноида в подходящей категории функторов, оснащенной D-сверткой (Day convolution), эквивалентно предыдущим.

Аппликатив как слабый замкнутый функтор

Стандартное определение аппликативного функтора на Haskell имеет вид:

```
class Functor f => Applicative f where
  (<*>) :: f (a -> b) -> (f a -> f b)
  pure  :: a -> f a
```

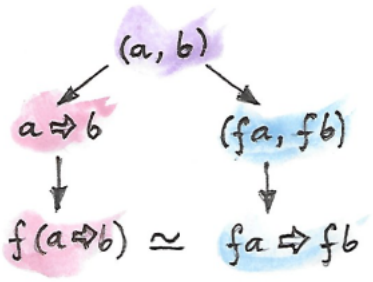
На первый взгляд кажется, что здесь нет моноидальной структуры. Это больше похоже на сохранение стрелок функций (я добавил несколько лишних скобок, чтобы предложить такую интерпретацию).

Категорно, функторы, «сохраняющие стрелки», известны как замкнутые функторы. Рассмотрим определение замкнутого функтора f между двумя категориями $\mathcal{C}$ и $\mathcal{D}$. Можно предположить, что обе категории замкнуты, то есть у них имеются внутренние hom-объекты для каждой пары объектов. Внутренние hom-объекты также называются функциональными объектами или экспонентами. Обычно они определяются через правое сопряжение к функтору произведения:

$$\mathcal{C}(z \times a, b) \cong \mathcal{C}(z, a \Rightarrow b)$$

Чтобы различать множества морфизмов и функциональные объекты (в **Set** это одно и то же), временно будем использовать двойные стрелки для функциональных объектов.

Можно взять функтор f и подействовать с его помощью на функциональный объект $a \Rightarrow b$ в категории $\mathcal{C}$, получая объект $f(a \Rightarrow b)$ в $\mathcal{D}$. Или, можно отобразить два объекта, a и b из $\mathcal{C}$, к $\mathcal{D}$, а затем построить функциональный объект в $\mathcal{D}$: $f\,a \Rightarrow f\,b$.



Функтор называется замкнутым, если эти два результата изоморфны (две стрелки помечены категориями, в которых они определены):

$$f(a \Rightarrow_{\mathcal{C}} b) \cong (f\,a \Rightarrow_{\mathcal{D}} f\,b)$$

и если этот функтор сохраняет единичный объект:

$$i_{\mathcal{D}} \cong f\,i_{\mathcal{C}}$$

Что представляет собой единичный объект? Обычно, это единица того же произведения, которая использовалась для определения функционального объекта с помощью сопряжения. Употреблено слово «обычно», потому что можно определить замкнутую категорию и без произведения.

Заметим, что две стрелки и два объекта определены для двух разных произведений. Первый изоморфизм должен быть естественным, как в a, так и в b. Также, для полноты картины, имеются несколько диаграмм, которые должны быть коммутативными.

Два изоморфизма, определяющие замкнутый функтор, можно ослабить и заменить однонаправленными морфизмами. В результате, получим слабый замкнутый функтор:

$$f(a \Rightarrow b) \rightarrow (fa \Rightarrow fb)$$
$$i \rightarrow f\,i$$

Это почти похоже на определение аппликатива, за исключением одной проблемы: как можно восстановить естественное преобразование, которое мы называем чистым (pure), из одного морфизма $i \to f\, i$.

Один из способов сделать это – с позиции прочности. Эндофунктор f имеет тензорную прочность, если существует естественное преобразование:

$$\mathsf{st}_{c\,a} :: c \otimes f\, a \to f(c \otimes a)$$

Будем думать о c как о контексте, в котором выполняется вычисление $f\, a$. Прочность означает, что можно использовать этот внешний контекст внутри вычислений.

В категории **Set**, где тензорное произведение заменено на декартово, все функторы имеют каноническую прочность. На Haskell это можно определить как:

```
st (c, fa) = fmap ((,) c) fa
```

Морфизм в определении слабого замкнутого функтора переводится к:

```
unit :: () -> f ()
```

Notice that, up to isomorphism, the unit type `()` is the unit with respect to cartesian product. The relevant isomorphisms are:

$$\lambda_a :: ((), a) \to a$$
$$\rho_a :: (a, ()) \to a$$

Это вывод от Rivas и Jaskelioff [3] («Представления о вычислениях как моноидах»):

$a \cong (a, ())$	закон единицы, ρ^{-1}
$\to (a, f())$	слабая единица
$\to f(a, ())$	прочность
$\cong fa$	закон поднятой единицы, $f\, \rho$

[3] https://arxiv.org/pdf/1406.4823v1.pdf

Прочность необходима, если начинать со слабого замкнутого (или моноидального — см. следующий раздел) эндофунтора в произвольной замкнутой (или моноидальной) категории и требуется получить чистый результат внутри этой категории, но не после того, как вы ограничите ее на **Set**.

Однако, существует альтернативный вывод с использованием леммы Йонеды:

$$\begin{aligned} f() &\cong \text{forall } a.\, (() \to a) \to f\, a && \text{Йонеда} \\ &\cong \text{forall } a.\, a \to f\, a && \text{поскольку: } (() \to a) \cong a \end{aligned}$$

Мы восстанавливаем всю естественную трансформацию из одного значения. Преимущество этого вывода в том, что он распространяется за пределы эндофункторов и не требует прочности. Как мы увидим позже, это также хорошо согласуется с определением аппликативности, которое дает D-свертка. Лемма Йонеды работает только для функторов со значениями в **Set**, но то же самое можно сказать и о D-свертке (существуют расширенные версии как Йонеды, так и D-свертки, но они не будут здесь обсуждаться).

Можно определить категорную версию аппликативного функтора Haskell как *слабый замкнутый функтор*, идущий от замкнутой категории $\mathcal{C}$ к **Set**. Это функтор, оснащенный естественным преобразованием:

$$f(a \Rightarrow b) \to (f\, a \to f\, b)$$

где $a \Rightarrow b$ — внутренний hom-объект в $\mathcal{C}$ (вторая стрелка — это тип функции в **Set**), а также функция:

$$1 \to f\, i$$

где 1 — одноэлементное множество, а i — единичный объект в $\mathcal{C}$.

Важность категорного определения состоит в том, что оно сопровождается дополнительными тождественностями или «аксиомами». Слабый замкнутый функтор должен быть совместим со структурой обеих категорий. Мы не будем здесь вдаваться в подробности, потому что, на самом деле, нас интересуют только моноидальные замкнутые категории, в которых эти аксиомы легче выразить.

Определение слабых замкнутых функторов легко переводится на Haskell:

```
class Functor f => Closed f where
  (<*>) :: f (a -> b) -> f a -> f b
  unit  :: f ()
```

Аппликатив как слабый моноидальный функтор

Несмотря на то, что можно определить замкнутую категорию без моноидальной структуры, на практике обычно работа ведется с моноидальными категориями. Это отражено в эквивалентном определении аппликативного функтора Haskell как слабого моноидального функтора. На Haskell можно было бы записать:

```
class Functor f => Monoidal f where
  (>*<) :: (f a, f b) -> f (a, b)
  unit  :: f ()
```

Это определение эквивалентно предыдущему определению замкнутого функтора. Это потому, что, как мы видели, функциональный объект в моноидальной категории определяется в терминах произведения. Можно показать эту эквивалентность в более общей категорной постановке.

На этот раз, начнем с симметричной замкнутой моноидальной категории $\mathcal{C}$, в которой функциональный объект определяется через правого сопряженного к тензорному произведению:

$$C(z \otimes a, b) \cong C(z, a \Rightarrow b)$$

Как обычно, тензорное произведение ассоциативно и унитально — с единичным объектом i — с точностью до изоморфизма. Симметрия определяется естественным изоморфизмом:

$$\gamma :: a \otimes b \to b \otimes a$$

Функтор f между двумя моноидальными категориями является слабым моноидальным, если существуют: (1) естественное преобразование

$$f\,a \otimes f\,b \to f(a \otimes b)$$

и (2) морфизм

$$i \to f\,i$$

Отметим, что произведения и единицы по обе стороны от двух отображений относятся к разным категориям.

(Слабый) моноидальный функтор также должен сохранять законы ассоциативности и единицы.

Например, тройное произведение

$$f\,a \otimes (f\,b \otimes f\,c)$$

может быть преобразовано с помощью ассоциатора α, для получения

$$(f\,a \otimes f\,b) \otimes f\,c$$

затем преобразовано к

$$f(a \otimes b) \otimes f\,c$$

а далее к

$$f((a \otimes b) \otimes c)$$

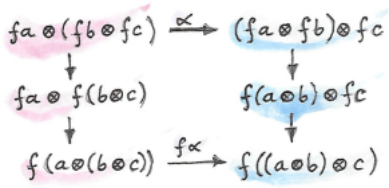
Или его можно сначала преобразовать в

$$fa \otimes f(b \otimes c)$$

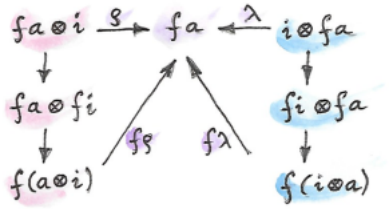
а затем, к

$$f(a \otimes (b \otimes c))$$

Эти два значения должны быть эквивалентны, относительно ассоциатора, в $\mathcal{C}$.



Аналогично, $f\,a \otimes i$ можно упростить до $f\,a$, используя правый унитор ρ в $\mathcal{D}$. Или, его можно сначала преобразовать в $f\,a \otimes f\,i$, затем в $f(a \otimes i)$, а затем в $f\,a$, используя правый унитор в $\mathcal{C}$. Оба пути должны быть эквивалентны (аналогично для левой идентичности).



Теперь рассмотрим функторы от $\mathcal{C}$ к **Set**, где **Set** оснащена обычным декартовым произведением и одноэлементным множеством в качестве единицы. *Слабый моноидальный функтор* определяется: (1) естественным преобразованием:

$$(f\,a, f\,b) \to f(a \otimes b)$$

и (2) выбором элемента множества $f\,i$ (функция от 1 к $f\,i$ выбирает элемент из этого множества).

Требуется, чтобы целевой категорией была **Set**, потому что мы хотим иметь возможность использовать лемму Йонеды, чтобы показать эквивалентность стандартному определению аппликатива. Я вернусь к этому моменту позже.

Эквивалентность

Определения слабого замкнутого и слабого моноидального функтора эквивалентны, когда $\mathcal{C}$ — замкнутая симметрическая моноидальная кате-

гория. Доказательство опирается на существование сопряжения, в частности, единицы и ко-единицы сопряжения:

$$\eta_a :: a \to (b \Rightarrow (a \otimes b))$$
$$\varepsilon_b :: (a \Rightarrow b) \otimes a \to b$$

Например, предположим, что f является слабой замкнутой. Требуется построить отображение

$$(f\, a, f\, b) \to f(a \otimes b)$$

Сначала применим поднятую пару (единица, тождественность), $(f\, \eta, f_{\mathsf{id}})$,

$$(f\, a \to f(b \Rightarrow a \otimes b), f_{\mathsf{id}})$$

к левой стороне. Получаем:

$$(f(b \Rightarrow a \otimes b), f\, b)$$

Теперь можно использовать слабый замкнутый морфизм (в некаррированной форме):

$$(f(b \Rightarrow x), f\, b) \to f\, x$$

получая:

$$f(a \otimes b)$$

И наоборот, предполагая свойство слабой моноидальности, можно показать, что функтор является слабым замкнутым, то есть реализовать следующую функцию:

$$(f(a \Rightarrow b), f\, a) \to f\, b$$

Сначала используем слабый моноидальный морфизм в левой части:

$$f((a \Rightarrow b) \otimes a)$$

а затем используем ко-единицу (она же, оценочный морфизм), чтобы получить желаемый результат, $f\, b$.

Существует еще одно представление аппликативов с использованием D-свертки. Но прежде чем заниматься этим, нужно немного освежить знания в области исчисления.

Исчисление концов

Концы и коконцы — очень полезные конструкции, обобщающие пределы и копределы. Они определяются через универсальные конструкции [4]. У них есть несколько фундаментальных свойств, которые снова и снова используются в категорных вычислениях. Я лишь введу обозначения и несколько важных тождеств. Мы будем работать с симметричной моноидальной категорией $\mathcal{C}$ с функторами от $\mathcal{C}$ к $\mathbf{Set}$ и профункторами от $\mathcal{C}^{\mathsf{op}} \times \mathcal{C}$ к $\mathbf{Set}$. Конец профунтора [5] p представляет собой множество, обозначаемое в виде

$$\int_a p\, a\, a$$

Самое важное в концах то, что множество естественных преобразований между двумя функторами f и g можно представить как конец:

$$[\mathcal{C}, \mathbf{Set}](f, g) = \int_a \mathcal{C}(f\, a, g\, a)$$

В Haskell конец соответствует универсальному количественному определению функтора смешанной дисперсии. Например, формула естественного преобразования принимает знакомую форму:

```
forall a. f a -> g a
```

[4] https://bartoszmilewski.com/2014/07/15/natural-transformations-and-ends/
[5] https://bartoszmilewski.com/2016/07/25/profunctors-as-relations/

Лемму Йонеды, касающуюся естественных преобразований, также можно записать с использованием конца:

$$\int_z (\mathcal{C}(a, z) \to f\ z) \cong f\ a$$

На Haskell это можно записать как эквивалентность:

```
forall z. ((a -> z) -> f z) ≅ f a
```

которое является обобщением преобразования передачи продолжения.

Двойственное понятие ко-конца аналогично записывается с использованием символа интеграла, с «переменной интегрирования» в верхнем индексе:

$$\int^a p\ a\ a$$

На псевдо-Haskell ко-конец представляется квантором существования. В Haskell можно определить экзистенциальные типы данных, преобразовав экзистенциальную квантификацию в универсальную. Соответствующая тождественность с точки зрения ко-концов и концов имеет вид:

$$(\int^z p\ z\ z) \to y \cong \int_z (p\ z\ z \to y)$$

На псевдо-Haskell эта формула используется для преобразования функций, принимающих экзистенциальные типы, в полиморфные функции:

```
(exists z. p z z) -> y ≅
                        forall z. (p z z -> y)
```

Интуитивно, это имеет смысл. Если имеется необходимость определить функцию, принимающую экзистенциальный тип, то должна существовать возможность обрабатывать любой тип.

Эквивалент леммы Йонеды для ко-концов:

$$\int^z f\, z \times \mathcal{C}(z, a) \cong f\, a,$$

или, на псевдо-Haskell:

```
exists z. (f z, z -> a) ≅ f a
```

(интуитивно понятно, что единственное, что вы можете сделать с этой парой, — это отобразить функцию над первым компонентом).

Существует также контрвариантная версия:

$$\int^z \mathcal{C}(a, z) \times f\, z \cong f\, a$$

где f — контравариантный функтор (он же, предпучок). На псевдо-Haskell:

```
exists z. (a -> z, f z) ≅ f a
```

(интуитивно понятно, что единственное, что можно сделать с этой парой, — это применить контр-отображение первого компонента ко второму компоненту).

Используя ко-концы, можно определить тензорное произведение в категории функторов $[\mathcal{C}, \mathbf{Set}]$. Это произведение называется D-сверткой:

$$(f \star g)\, a = \int^{x\, y} f\, x \times g\, y \times \mathcal{C}(x \otimes y, a)$$

Это бифунктор в этой категории (т.е., его можно использовать для отмены естественных преобразований). Он ассоциативен и симметричен, с точностью до изоморфизма. У него также есть единица — $\hom$-функтор $\mathcal{C}(i, -)$, где i — моноидальная единица в $\mathcal{C}$. Другими словами, D-свертка наделяет категорию $[\mathcal{C}, \mathbf{Set}]$ моноидальной структурой.

Проверим законы единицы.

$$(\mathcal{C}(i,-) \star g)\, a = \int^{x\, y} \mathcal{C}(i,x) \times g\, y \times \mathcal{C}(x \otimes y, a)$$

Можно использовать контравариант Йонеды для «интегрирования по x», получая:

$$\int^{y} g\, y \times \mathcal{C}(i \otimes y, a)$$

Учитывая, что i — единица тензорного произведения в $\mathcal{C}$, получаем:

$$\int^{y} g\, y \times \mathcal{C}(y, a)$$

Ковариант Йонеды позволяет «интегрировать по y», получая желаемое $g\, a$. Тот же метод работает для закона правой единицы.

Аппликатив как моноид

Для моноидальной категории, всегда можно определить моноид как объект m, снабженный двумя морфизмами:

$$\mu :: m \otimes m \to m$$
$$\eta :: i \to m$$

удовлетворяющие законам ассоциативности и унитальности.

Мы показали, что категория функторов $[\mathcal{C}, \mathbf{Set}]$ (где $\mathcal{C}$ — симметричная моноидальная категория) моноидальна относительно D-свертки. Объектом этой категории является функтор f. Два морфизма, которые сделали бы его кандидатом на роль моноида, являются естественными преобразованиями:

$$\mu :: f \star f \to f$$
$$\eta :: C(i,-) \to f$$

Компонент естественного преобразования μ можно переписать как

$$\left(\int^{x\,y} f\,x \times f\,y \times C(x \otimes y, a)\right) \to f\,a$$

который эквивалентен

$$\int_{x\,y} (f\,x \times f\,y \times C(x \otimes y, a) \to f\,a)$$

или, при каррировании:

$$\int_{x\,y} (f\,x, f\,y) \to C(x \otimes y, a) \to f\,a$$

Оказывается, что определенный таким образом моноид эквивалентен слабому моноидальному функтору. Это показали Rivas и Jaskelioff. Следующий вывод принадлежит Bob Atkey.

Хитрость заключается в том, чтобы начать со всего множества естественных преобразований, от $f \star f$ к f. Умножение μ — лишь один из них. Множество естественных преобразований выразим как конец:

$$\int_a ((f \star f)\,a \to f\,a)$$

Подставив выражение для компонента μ, получим:

$$\int_{a\,x\,y} (f\,x, f\,y) \to \mathcal{C}(x \otimes y, a) \to f\,a$$

Конец над a не включает первый аргумент, поэтому можно переместить знак интеграла:

$$\int_{x\,y} (f\,x, f\,y) \to \int_a \mathcal{C}(x \otimes y, a) \to f\,a$$

Затем используем лемму Йонеды, чтобы «выполнить интегрирование» по a:

$$\int_{x\,y} (f\,x, f\,y) \to f(x \otimes y)$$

Это можно распознать это как множество естественных преобразований, определяющих слабый моноидальный функтор. Мы установили взаимно однозначное соответствие между этими естественными преобразованиями и преобразованиями, определяющими моноидальное умножение с использованием D-свертки.

Оставшиеся действия — показать эквивалентность единицы относительно D-свертки и второй части определения слабого моноидального функтора — морфизма:

$$1 \to f\,i$$

Начнем с множества естественных преобразований, которое содержит η:

$$\int_a (i \to a) \to f\,a$$

С помощью Йонеды, это просто $f\,i$. Выбор элемента из множества эквивалентен определению морфизма из одноэлементного множества 1, поэтому, для любого выбора η, получаем:

$$1 \to f\,i$$

и наоборот. Оба определения эквивалентны.

Заметим, что моноидальная единица η при D-свертке становится определением pure в аппликативной версии Haskell. Действительно, когда категория $\mathcal{C}$ заменяется на $\mathbf{Set}$, f становится эндофунктором, а единица D-свертки, $\mathcal{C}(i, -)$, становится тождественным функтором Id. Получаем:

$$\eta :: \mathsf{Id} \to f$$

или, в компонентах:

$$\mathsf{pure} :: a \to f\,a$$

Итак, строго говоря, определение аппликатива в Haskell смешивает элементы слабого замкнутого функтора и моноидальной единицы при D-свертке.

Благодарности

Я благодарен Mauro Jaskelioff и Exequiel Rivas за переписку, а также Bob Atkey, Dimitri Chikhladze и Make Shulman за ответы на мои вопросы по Math Overflow.