Свободные теоремы для функтора. Конспект.

2 апреля 2019 г.

Аннотация

Как известно, для того, чтобы некоторый типовой оператор `F` с кайндом `*->*` был объявлен представителем класса `Functor`, должна быть определена функция

`fmap :: a -> b -> Fa -> Fb`

, для которой должны выполняться следующие два закона:

```
fmap id = id
```

```
fmap (f . g) = (fmap f) . (fmap g)
```

Оказывается, второй закон не нужен, потому что следует из первого.

1 Формализм

1.1 System F

Будем работать в System F. Что у нас есть кроме просто типизированного лямбда исчисления: абстракция (Λ) терма по типу и аппликация (`[]`) типа к терму, соответственно, появляются полиморфные типы (добавляется $\forall$). Естественным образом добавляются правила эвалюации (β-редукции) и утверждения о типизации.

Таким образом, операций над типами у нас две: $\to$ и $\forall$

1.2 Frame semantics и новые обозначения

Далее будет активно использоваться формализм, описанный в статье Водлера (https://people.mpi-sws.org/ dreyer/tor/papers/wadler.pdf). Оттуда будут позаимствованы обозначения U, D_X, $[D_A \to D_B]$ и другие.

Кроме того, введём для сокращения формул следующие переобозначения: функцию $\varphi_{A,B} : D_{A\to B} \to [D_A \to D_B]$, переводящую значение функционального типа в реальную функцию, обозначим через черту сверху, параметры A и B выводятся неявно. Таким же образом (чертой сверху) обозначим и Φ. Чертой снизу обозначим функции ψ и Ψ.

$$\bar{x} := \varphi(x)_{A,B}$$

$$\bar{y} := \Phi(y)_{A,B}$$

$$\underline{x} := \psi(x)_{A,B}$$

$$\underline{y} := \Psi(y)_{A,B}$$

При этом переобозначении не возникает неоднозначности, потому что то, какая именно функция подразумевается, определяется по аргументу.

Для краткости будем опускать окружение после семантических скобок, если оно пустое (в частности, если контекст пустой, то путое окружение с ним согласуется):

$$[\![T]\!] := [\![T]\!]\overline{\mathcal{A}}$$

1.3 Что мы можем выразить?

Достаточно ли System F, чтобы выразить всё, что нужно? Оказывается, да. Рекурсивные типы выражаются в $SystemF$ следующим образом:

```
Lfix X. F X = All X. (F X -> X) -> X
```

Доказательство можно найти здесь: http://homepages.inf.ed.ac.uk/wadler/papers/free-rectypes/free-rectypes.txt

Также в System F можно выразить сумму и произведение типов.

Таким образом, любой нужный нам функтор выразим в System F. (?)

1.4 Типы с дыркой

Минуточку, а что такое функтор? В System F нет такого понятия, как оператор над типами, нет кайндов. Всё это появляется в System F_ω. Но для простоты мы можем искусственно ввести понятие функтора в System F как некоторого типа с дыркой (_). Дырка играет роль места, на которое должен встать тип при аппликации к нему функтора.

Для того, чтобы можно было определить `fmap`, дырка должна находиться в ковариантной позиции.

Определим множество функторов параллельно с множеством кофункторов следующим образом (нотация из TAPL):

$$
\begin{array}{l}
Func ::= \\
\quad \_ \\
\quad T \\
\quad \forall X.Func \\
\quad Cofunc \to Func
\end{array}
\qquad (1) \qquad
\begin{array}{l}
Cofunc ::= \\
\quad T \\
\quad \forall X.Func \\
\quad Func \to Cofunc
\end{array}
\qquad (2)
$$

Здесь T — это тип, X — типовая переменная.

По сути, функтор — это тип с дырками, у которого все дырки находятся в ковариантных позициях, кофунктор — напротив, содержит дырки только в контрвариантных позициях.

Пусть F принадлежит множеству функторов или кофункторов. Введём обозначение $F[X] ::= [\_ \to X]F$. То есть $F[X]$ — это тип, который получается при подстановке в F вместо дырки типовой переменной X.

1.5 Что мы доказываем?

Запишем то, что хотим доказать, в новых обозначениях. Зафиксируем некоторую frame model. Пусть F — это функтор, $fmap$ — это терм в System F, при чём

```
⊢ fmap : ∀A.∀B . (A→B)→F[A]→F[B]
```

Кроме того, выполняется первый закон, то есть $\forall A \in U$

$$\overline{\overline{\overline{\overline{\overline{[\![fmap_G]\!]}AA[\![id_A]\!]}}}} = \overline{[\![id_{F[A]}]\!]}$$

где $id_X := \lambda x : X.x$ (в принципе, ничего не мешает записать здесь полиморфный id). Равенство — это просто равенство функций.

Тогда выполняется второй закон, то есть $forall A, B, C \in U$

2 Естественный fmap

2.1 Построение

По **функтору** F можно построить терм $\texttt{nmap}_F :: \forall X.\forall Y.(X \to Y) \to F[X] \to F[Y]$, удовлетворяющий законам `fmap`.

По **кофунктору** F можно построить терм $\texttt{conmap}_F :: \forall X.\forall Y.(X \to Y) \to F[Y] \to F[X]$, удовлетворяющий законам `cofmap`.

Построим эти термы параллельно.

```
nmap_F =
    | F == _      = ΛX.ΛY . λφ: X→Y . φ                                          : (X→Y)→X→Y
    | F == T      = ΛX.ΛY . λφ: X→Y . id_T                                       : (X→Y)→T→T
    | F == ∀C.G   = ΛX.ΛY . λφ: X→Y . λg: (∀C.G[X]) . ΛC . nmap_G φ g[C]         : (X→Y)→∀C.G[X]→∀C.G[Y]
    | F == Q → G  = ΛX.ΛY . λφ: X→Y . λψ: Q[X] → G[X] . λq: Q[Y] . nmap_G φ (ψ (conmap_Q φ q))
                  : (X→Y)→(Q[X]→G[X])→(Q[Y]→G[Y])
```

```
conmap_F =
    | F == T      = ΛX.ΛY . λφ: X→Y . id_T                                       : (X→Y)→T→T
    | F == ∀C.G   = ΛX.ΛY . λφ: X→Y . λq: (∀C.Q[Y]) . ΛC . conmap_Q φ q[C]       : (X→Y)→∀C.G[Y]→∀C.G[X]
    | F == G → Q  = ΛX.ΛY . λφ: X→Y . λψ: G[Y]→Q[Y] . λg: G[X] . conmap_Q φ(ψ(nmap_G φ g))
                  : (X→Y)→(G[Y]→Q[Y])→(G[X]→Q[X])
```

Возможность построить эти функции универсальным образом гарантирована ковариантностью (и контрвариантностью в случае кофунктора) позиций дырок.

Здесь G обозначает функтор, Q — кофунктор, T — обычный тип без дырки.

2.2 Законы для nmap

Имеет место следующие утверждения:

$$\overline{\overline{\overline{\overline{[\![nmap_G]\!]}AA[\![id_A]\!]}}} = \overline{[\![id_{F[A]}]\!]}$$

$$\overline{\overline{\overline{\overline{[\![conmap_Q]\!]}AA[\![id_A]\!]}}} = \overline{[\![id_{Q[A]}]\!]}$$

Под равенством имеется в виду экстенсиональное равенство, $id := \Lambda X.\lambda x : X.x$

Его легко доказать параллельной (для функтора и кофунктора) индукцией по высоте дерева построения функтора (и кофунктора).

Аналогично можно доказать выполнение второго закона, но это даже не обязательно.

3 Свободная теорема

3.1 Формулировка

Пусть F — это функтор.

Пусть $fmap$ — это некоторый терм в System F, при чём

```
⊢ fmap : ∀A.∀B . (A→B)→F[A]→F[B]
```

Пусть есть некоторые термы p, q, f, g и типы A, B, A', B'

```
⊢ p : A  → B
⊢ q : A' → B'
⊢ f : A  → A'
⊢ g : B  → B'
```

При чём

```
g . p = q . f
```

Тогда имеет место следующее равенство:

```
(nmap_F g) . (fmap p) = (fmap q) . (nmap_F f)
```

Для наглядности здесь полезно нарисовать коммутативную диаграмму.

3.2 Доказательство

4 Итог

Пусть `fmap` — терм как в предыдущем разделе, при чём такой, что для любых типов A и B

```
fmap id_A = id_B
```

Тогда

```
fmap f = nmap id . fmap f = fmap id . nmap f = nmap f
```

Первое равенство объясняется тем, что `nmap id = id`, Второе раветство — применение свободной теоремы. Третье равенство следует из `fmap id = id`.

Теперь можно воспользоваться тем, что второй закон верен для *nmap*, а можно даже не пользоваться:

```
fmap f . fmap g = nmap f . fmap g = fmap id . nmap (f . g) = fmap (f . g)
```