Теория типов и некоторые её применения

Введение

Интересный пример того, как чистая (абстрактная) математика постепенно эволюционирует, приходя к содержательной практике.

- Кризис оснований математики, попытка переосмыслить логику.
- Лямбда-исчисление, применение интуиционистской логики к программированию.
- Инструменты, которыми широко пользуются.

Кризис оснований математики

- Кризис оснований математики начала XX века: определения приводят к неочевидным противоречиям.
- Пусть $S := \{\ x \mid x \notin x\ \}$, верно ли $S \in S$?
  «На некотором острове живёт брадобрей, который бреет всех, кто не бреется сам. Бреется ли сам брадобрей?»
- Может, наш стиль логического рассуждения вообще некорректен?
- Одна из попыток переосмысления — интуиционистская математика (Лёйтзен Эгберт Ян Брауэр, 1904 год).

BHK-интерпретация

- BHK-интерпретация (Лёйтзен Эгберт Ян Брауэр, Аренд Гейтинг, Андрей Николаевич Колмогоров) — интерпретация логических выражений. Обозначим переменными конструкции, которые мы умеем строить. Тогда:
  - умеем строить $\alpha \,\&\, \beta$, если умеем строить и α и β;
  - умеем строить $\alpha \vee \beta$, если умеем строить α или β, и мы знаем, что именно;
  - умеем строить $\alpha \to \beta$, если умеем перестраивать α в β;
  - $\perp$ не имеет построения.
- Любая интуиционистская формула доказуема в классической логике, но не наоборот.
- Подробнее про парадоксы и попытки их преодоления можно прочесть, например, у Стефана Клини в книге «Введение в метаматематику» (русский перевод 1957 года)

BHK интерпретация и программы

- BHK интерпретация и типы:
  - $\alpha \,\&\, \beta$ — тип упорядоченной пары значений;
  - $\alpha \vee \beta$ — алгебраический тип (тип-сумма);
  - $\alpha \to \beta$ — тип функций из α в β;
  - $\perp$ — значение, не имеющее построения (например, результат исключения).
- Доказательство — значение, *обитающее* в указанном типе. Вот и давайте подоказываем:

$$\vdash 3 : \texttt{int}$$

$$\vdash \texttt{fun x -> x+1} : \texttt{int} \to \texttt{int}$$

- Изоморфизм Карри-Ховарда

Программа (λ-исчисление)	Исчисление высказываний
Выражение	доказательство
Тип выражения	высказывание
Тип функции	импликация

Давайте ещё что-нибудь докажем

- Например, такая формула из ИИВ:

$$(A \lor B \to C) \to ((A \to C) \,\&\, (B \to C))$$

- За ней стоит такая интуиция:

$$c^{a+b} = c^a \cdot c^b$$

- И такое доказательство:

```
let proof f = (fun a -> f (Left a), fun b -> f (Right b))
```

- Почему это доказательство? Поскольку обитаемость эквивалентна наличию доказательства (согласно изоморфизму Карри-Ховарда)

Пример недоказуемого утверждения

«Если вчера шёл дождь, то сегодня будет солнечный день ИЛИ если сегодня будет солнечный день, то вчера шёл дождь»

$$(A \to B) \vee (B \to A)$$

- Это доказуемо в классической логике:

A	B	$A \to B$	$B \to A$	$(A \to B) \vee (B \to A)$
Л	Л	И	И	И
Л	И	Л	И	И
И	Л	И	Л	И
И	И	И	И	И

- Но недоказуемо в интуиционистской логике.
  Как получить из дождя вчера солнечный день сегодня?

Две основные идеи

- Давайте изучать языки программирования при помощи логических исчислений: функциональные языки (ИИП второго порядка), языки с зависимыми типами (ИИП первого порядка), линейные/уникальные типы (линейная логика), ...
- Давайте доказывать математические утверждения при помощи написания программ: интуиционистская теория типов (Coq, Agda, Lean), гомотопическая теория типов (Arend), ...

Изучаем языки: дженерики, шаблоны

- Полиморфные функции: `let id x = x`. Каков тип `id`? Вообще говоря, $\forall\alpha.\alpha \to \alpha$ (каков бы ни был тип, функция вернёт результат того же типа, который есть у аргумента).
- Дженерики:

```
template <class X>
class Z {
    X field;
    X field2;
}
```

  Что такое Z? Это функция, возвращающая тип по другому типу (генерик).
- Экзистенциальные типы: абстрагируют инкапсуляцию.
  John C. Mitchell, Gordon D. Plotkin, Abstract Types Have Existential Type
  http://homepages.inf.ed.ac.uk/gdp/publications/Abstract_existential.pdf

Изучаем языки: зависимые типы

`printf(char* format, ...)`

`sprintf "%s" : string -> string; sprintf "%d" : int -> string`

Язык Idris (вдохновлено https://github.com/mukeshtiwari/Idris/blob/master/Printf.idr):

```
data Format = FInt Format
            | FString Format
            | FOther Char Format
            | FEnd

f : List Char -> f
f ('%' :: 'd' :: cs ) = FInt ( f cs )
f ('%' :: 's' :: cs ) = FString ( f cs )
f ( c :: cs )         = FOther c ( f cs )
f []                  = FEnd

it : Format -> Type
it ( FInt f )      = Int -> it f
it ( FString f )   = String -> it f
it ( FOther _ f )  = it f
it FEnd            = String
```

```
fStr : String -> Format
fStr s = f ( unpack s )

toF : (fmt : Format) -> String -> it fmt
toF (FInt f) a      = \i => toF f (a ++ show i)
toF (FString f) a   = \s => toF f (a ++ s)
toF (FOther c f) a = toF f (a ++ singleton c)
toF FEnd a            = a

sprintf : ( s : String ) -> it ( fStr s )
sprintf s = toF ( fStr s ) ""

main : IO ()
main = putStrLn (sprintf "%s, %d" "a" (2*2))
```

Изучаем языки: линейные типы

Линейные и уникальные типы (Rust, Clean): гарантируют, что существует в точности один объект указанного типа. Данное изложение следует статье Филиппа Вадлера (Philip Wadler. A taste of linear logic)

$$\frac{[\Gamma] \vdash \alpha}{[\Gamma] \vdash !\alpha} \qquad \frac{\Gamma \vdash !\alpha \qquad \Delta, [\alpha] \vdash \beta}{\Gamma, \Delta \vdash \beta}$$

Линейная импликация:

$$\frac{\Gamma, \langle\alpha\rangle \vdash \beta}{\Gamma \vdash \alpha \multimap \beta} \qquad \frac{\Gamma \vdash \alpha \multimap \beta \qquad \Delta \vdash \alpha}{\Gamma, \Delta \vdash \beta}$$

Правила для связок: конъюнкция и дизъюнкция

Мультипликативная конъюнкция (возможно построить и α и β одновременно):

$$\frac{\Gamma \vdash \alpha \qquad \Delta \vdash \beta}{\Gamma, \Delta \vdash \alpha \otimes \beta} \qquad \frac{\Gamma \vdash \alpha \otimes \beta \qquad \Delta, \langle\alpha\rangle, \langle\beta\rangle \vdash \varphi}{\Gamma, \Delta \vdash \varphi}$$

Аддитивная конъюнкция (возможно построить α и возможно построить β, что-то одно по нашему выбору):

$$\frac{\Gamma \vdash \alpha \qquad \Gamma \vdash \beta}{\Gamma \vdash \alpha \& \beta} \qquad \frac{\Gamma \vdash \alpha \& \beta}{\Gamma \vdash \alpha} \qquad \frac{\Gamma \vdash \alpha \& \beta}{\Gamma \vdash \beta}$$

Аддитивная дизъюнкция (возможно построить α или возможно построить β, что-то одно по их выбору):

$$\frac{\Gamma \vdash \alpha}{\Gamma \vdash \alpha \oplus \beta} \qquad \frac{\Gamma \vdash \beta}{\Gamma \vdash \alpha \oplus \beta} \qquad \frac{\Gamma \vdash \alpha \oplus \beta \qquad \Delta, \langle\alpha\rangle \vdash \varphi \qquad \Delta, \langle\beta\rangle \vdash \varphi}{\Gamma, \Delta \vdash \varphi}$$

Изучаем языки: рекурсия, исключения и противоречивость

Рассмотрим программу:

```
let f x = f (x+1)
```

Каков тип `f`? Какой угодно, поскольку `f` не возвращает результата: «Из лжи следует, что угодно».

```
let ens_pos x = if x > 0 then x else failwith "x <= 0"
let ens_ne s = if String.length s > 0 then s else failwith "s = \"\""
```

Иными словами: тип результата `failwith` может быть любым: $\forall a.a$.
Таким образом, Тьюринг-полный язык обязательно противоречив (неограниченная рекурсия неизбежно позволяет доказать ложь).

Интуиционистская теория типов

- Пер Мартин-Лёф, 1972 год (An Intuitionistic Theory of Types), 1982 год (Constructive mathematics and computer programming).
- Примерно соответствует исчислениям предикатов первого и второго порядков (включает зависимые типы, Π и Σ типы и т.п.).
- Явно введённые в теорию индуктивные типы (расширение алгебраических типов) и W-типы.
- Иерархия универсумов.
- Специальный тип для равенства.

Доказательства (пример на языке Агда)

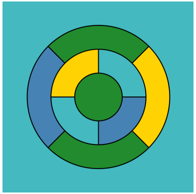
Подробнее про язык Агда тут:
https://wiki.portal.chalmers.se/agda/pmwiki.php

Равенство — тип, параметризованный значениями. Мы ожидаем, что $1 = 1$ обитаем (`refl : 1 = 1`), а $1 = 2$ — нет. Докажем ассоциативность сложения.

```
module example-proof where

open import Data.Nat using (ℕ; zero; suc; _+_)
open import Relation.Binary.PropositionalEquality using (_≡_; refl; cong)

+-assoc : Set
+-assoc = ∀ (x y z : ℕ) → x + (y + z) ≡ (x + y) + z

+-assoc-proof : ∀ (x y z : ℕ) → x + (y + z) ≡ (x + y) + z

+-assoc-proof zero y z = refl
+-assoc-proof (suc x) y z = cong suc (+-assoc-proof x y z)
```

Формализация математики: задача о четырёх красках

- Раскрасить карту так, чтобы никакие две страны с общей границей не имели один цвет.



- Для трёх красок не решается. Для пяти красок — доказать легко.
- Для четырёх (1976 год, Кеннет Аппель и Вольфганг Хакен) — сводится к разбору 1834 случаев.
  - Перебор требовал компьютера, вручную не проверить.
  - Цифру уменьшили до 633 случаев, но всё равно слишком много.
  - Компьютерная программа может содержать ошибки — многие не верят.
  - В 2005 году Джордж Гонтир доказал корректность перебора с помощью Coq. https://www2.tcs.ifi.lmu.de/~abel/lehre/WS07-08/CAFR/4colproof.pdf

Гомотопическая теория типов

Много технически сложных, но элементарных доказательств. Много неэлементарых, но простых. Математики должны иметь возможность создавать сложные и неэлементарные доказательства.

Гомотопическая теория типов (Владимир Александрович Воеводский, 2006 год).
https://homotopytypetheory.org/book/
https://arend-lang.github.io/

Логика	λ-исчисление	Топология
Высказывание	Тип	Пространство
Доказательство	Терм	Точка в пространстве
Предикат	Зависимый тип	Расслоение
Равенство	*Выделенный* зависимый тип	Пространство путей
Доказательство равенства	Элемент равенства	Путь между точками

Будем считать два значения a и b в пространстве X равными, если они связаны путём: непрерывной функцией $f : I \to X$, где $I = [0, 1]$, причём $f(0) = a$, $f(1) = b$.

Верификация кода

- Не всё проверяется тестированием: можно ли доказать свойства про код, не запуская его?
- Много подходов: SMT, model checking...
- Но сам по себе изоморфизм Карри-Ховарда подсказывает прямой способ — тип и есть часть спецификации кода. Может, её можно расширить?
- Coq (Rocq) — разработан INRIA и др., 1989 год. https://rocq-prover.org/
    - Язык с зависимыми типами данных, предназначенный для записи и проверки доказательств — доказана обширная библиотека математических фактов.
    - Gallina — функциональный язык программирования (не Тьюринг-полный).
    - Выделение кода из доказательств.

Верификация кода: зависимые типы

- CompCert — верифицированный компилятор C, генерирующий код, доказуемо эквивалентный исходному.
  https://compcert.org/
  Semantic preservation theorem: For all source programs S and compiler-generated code C, if the compiler, applied to the source S, produces the code C, without reporting a compile-time error, then the observable behavior of C is one of the possible observable behaviors of S.
  Вместе с кодом компилятор генерирует доказательство сохранения семантики (в данный момент — сохранение семантики синтаксического дерева исходного кода и ассемблера).
- Доказательство свойств про код — можно какой-то другой код транслировать в Coq.
  https://github.com/formal-land/coq-of-solidity
- Certified Programming with Dependent Types: A Pragmatic Introduction to the Coq Proof Assistant (Adam Chipala).

Верификация кода: уточнённые типы

Уточнённый тип — тип вида $\{\tau(x) \mid P(x)\}$, где P — некоторый предикат.
Пример на LiquidHaskell (https://wiki.haskell.org/Liquid_Haskell):

```
data [a] <p :: a -> a -> Prop> where
    | []  :: [a] <p>
    | (:) :: h:a -> [a<p h>]<p> -> [a]<p>
```

- `h:a` — голова (h) имеет тип a
- `[a<p h>]<p>` — хвост состоит из значений типа a, уточнённых p — $\{t : a \mid p\ h\ t\}$ (карринг: `a <p h>`).

```
{-@ type IncrList a = [a] <{\xi xj -> xi <= xj}> @-}
{-@ insertSort    :: (Ord a) => xs:[a] -> (IncrList a) @-}
insertSort []     = []
insertSort (x:xs) = insert x (insertSort xs)
```

Похожий проект есть для ML: F* (https://fstar-lang.org/)