

# ТЕОРЕТИЧЕСКИЕ ДОМАШНИЕ ЗАДАНИЯ

Теория типов, ИТМО, совместно М3232-М3239 и М3332-М3339, весна 2024 года

## Домашнее задание №1: лямбда исчисление — бестиповое и просто-типизированное

1. Бесконечное количество комбинаторов неподвижной точки. Дадим следующие определения

$$\begin{aligned} L &:= \lambda abcdefghijklmnopqrstuvwxyzr.(thisisafixedpointcombinator) \\ R &:= LLLLLLLLLLLLLLLLLLLLLLLLLLLLL \end{aligned}$$

В данном определении терм  $R$  является комбинатором неподвижной точки: каков бы ни был терм  $F$ , выполнено  $R F =_{\beta} F (R F)$ .

- (a) Докажите, что данный комбинатор — действительно комбинатор неподвижной точки.
- (b) Пусть в качестве имён переменных разрешены русские буквы. Постройте аналогичное выражение по-русски: с 33 параметрами и осмысленной русской фразой в терме  $L$ ; покажите, что оно является комбинатором неподвижной точки.
2. Найдите необитаемый тип в просто-типизированном лямбда-исчислении (напомним: тип  $\tau$  называется необитаемым, если ни для какого  $P$  не выполнено  $\vdash P : \tau$ ).
3. Напомним определение: комбинатор — лямбда-выражение без свободных переменных. Также напомним:

$$\begin{aligned} S &:= \lambda x.\lambda y.\lambda z.x z (y z) \\ K &:= \lambda x.\lambda y.x \\ I &:= \lambda x.x \end{aligned}$$

Известна теорема о том, что для любого комбинатора  $X$  можно найти выражение  $P$  (состоящее только из скобок, пробелов и комбинаторов  $S$  и  $K$ ), что  $X =_{\beta} P$ . Будем говорить, что комбинатор  $P$  *выражает* комбинатор  $X$  в базисе  $SK$ .

Косвенным аргументом (пояснением, но не доказательством!) в пользу этой теоремы являются два следующих соображения:

- теорема о замкнутости ИФИИВ: если  $\vdash \varphi$ , то  $\vdash_{\rightarrow} \varphi$ , значит, если выражение имеет тип, то этот тип можно получить с помощью доказательства в стиле Гильберта;
- типы комбинаторов  $S$  и  $K$  — это, соответственно, вторая и первая схемы аксиом.

Докажите тип следующих выражений как логическое высказывание с помощью гильбертового вывода и, пользуясь этим доказательством как источником вдохновения, выразите комбинаторы в базисе  $SK$ :

- (a)  $\lambda x.\lambda y.\lambda z.y$   
(b)  $\lambda x.\lambda y.\lambda z.yxz$   
(c)  $\bar{I}$   
(d)  $Not$   
(e)  $Xor$   
(f)  $InR$

4. Покажите на основании следующего преобразования полноту комбинаторного базиса  $SKI$  (проведите полное рассуждение по индукции, из которого будет следовать отсутствие в результате других термов, кроме  $SKI$ , бета-эквивалентность и определённость результата для всех комбинаторов  $\sigma$ ):

$$[\sigma] = \begin{cases} x, & \sigma = x \\ [\varphi] [\psi], & \sigma = \varphi \psi \\ K [\varphi], & \sigma = \lambda x.\varphi, \quad x \notin FV(\varphi) \\ I, & \sigma = \lambda x.x \\ [\lambda x. [\lambda y.\varphi]], & \sigma = \lambda x.\lambda y.\varphi, \quad x \in FV(\varphi), x \neq y \\ S [\lambda x.\varphi] [\lambda x.\psi], & \sigma = \lambda x.\varphi \psi, \quad x \in FV(\varphi) \cup FV(\psi) \end{cases}$$

Заметим, что данные равенства объясняют смысл названий комбинаторов:

$S$  verSchmelzung, «сплавнение»  
 $K$  Konstanz  
 $I$  Identität

5. Покажите, что следующая система комбинаторов образует полный базис в бестиповом лямбда-исчислении, но соответствующая им система аксиом в исчислении гильбертового типа не образует полного базиса для импликативного фрагмента:

$$\begin{aligned}
 I &:= \lambda x.x \\
 K &:= \lambda x.\lambda y.x \\
 S' &:= \lambda i.\lambda x.\lambda y.\lambda z.i \ (i \ ((x \ (i \ z)) \ (i \ (y \ (i \ z)))))
 \end{aligned}$$

Указание: покажите невыводимость  $(\varphi \rightarrow \varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \psi)$ .

6. Напомним определение аппликативного порядка редукции: редуцируется самый левый из самых вложенных редексов. Например, в случае выражения  $(\lambda x.I \ I) \ (\lambda y.I \ I)$  самые вложенные редексы — применения  $I \ I$ :

$$(\lambda x.\underline{I \ I}) \ (\lambda y.\underline{I \ I})$$

и надо выбрать самый левый из них:

$$(\lambda x.\underline{I \ I}) \ (\lambda y.I \ I)$$

- (a) Проведите аппликативную редукцию выражения 2 2.
- (b) Докажите или опровергните, что параллельная бета-редукция из теоремы Чёрча-Россера не медленнее (в смысле количества операций для приведения выражения к нормальной форме), чем нормальный порядок редукции с мемоизацией.
- (c) Найдите лямбда-выражение, которое редуцируется медленнее при нормальном порядке редукции, чем при аппликативном, даже при наличии мемоизации.
7. Напомним определение бета-редукции.  $A \rightarrow_\beta B$ , если:
- $A \equiv (\lambda x.P) \ Q$ ,  $B \equiv P \ [x := Q]$ , при условии свободы для подстановки;
  - $A \equiv (P \ Q)$ ,  $B \equiv (P' \ Q')$ , при этом  $P \rightarrow_\beta P'$  и  $Q = Q'$ , либо  $P = P'$  и  $Q \rightarrow_\beta Q'$ ;
  - $A \equiv (\lambda x.P)$ ,  $B \equiv (\lambda x.P')$ , и  $P \rightarrow_\beta P'$ .
- (a) Найдите лямбда-выражение, бета-редукция которого не может быть произведена из-за нарушения правила свободы для подстановки (для продолжения редукции потребуется производить переименование связанных переменных). Поясните, какое ожидаемое ценное свойство будет нарушено, если ограничение правила проигнорировать.
- (b) Покажите, что недостаточно наложить требования на исходное выражение, и свобода для подстановки может быть нарушена уже в процессе редукции исходно полностью корректного лямбда-выражения.
8. Будем говорить, что выражение  $A$  находится в *слабой заголовочной нормальной форме* (WHNF), если оно не имеет вид  $A \equiv (\lambda x.P) \ Q$  (то есть, самый верхний терм его не является редексом). Выражение находится в *заголовочной нормальной форме* (HNF), когда его верхний терм — не редекс и не лямбда-абстракция с бета-редексами в теле.
- (a) Приведите нормальным порядком редукции выражение 2 2 в СЗНФ.
- (b) Приведите нормальным порядком редукции выражение  $Y \ (\lambda f.\lambda x.(IsZero \ x) \ 1 \ (x \cdot f(x - 1))) \ 3$  в СЗНФ.
- (c) Верно ли, что «нормальность» формы выражения может в процессе редукции только усиливаться (никакая — слабая заголовочная Н.Ф. — заголовочная Н.Ф. — нормальная форма)?
9. Как мы уже разбирали,  $\not\vdash x \ x : \tau$  в силу дополнительных ограничений правила

$$\frac{}{\Gamma, x : \tau \vdash x : \tau} \quad x \notin FV(\Gamma)$$

Найдите лямбда-выражение  $N$ , что  $\not\vdash N : \tau$  в силу ограничений правила

$$\frac{\Gamma, x : \sigma \vdash N : \tau}{\Gamma \vdash \lambda x.N : \sigma \rightarrow \tau} \quad x \notin FV(\Gamma)$$

## Домашнее задание №2: задачи типизации лямбда исчисления

1. Рассмотрим подробнее отличия исчисления по Чёрчу и по Карри. Определим точно бета-редукцию в исчислении по Чёрчу:  $A \rightarrow_\beta B$ , если

$$\begin{aligned} A &= (\lambda x^\sigma. P) Q, & B &= P[x := Q] \\ A &= P Q, & B &= P Q' \text{ или } B = P' Q \text{ при } P \rightarrow_\beta P' \text{ и } Q \rightarrow_\beta Q' \\ A &= \lambda x^\sigma. P, & B &= \lambda x^\sigma. P' \text{ при } P \rightarrow_\beta P' \end{aligned}$$

- (a) Покажите, что в исчислении по Карри не выполняется даже «ограниченное» свойство распространения типизации (subject expansion): если  $\vdash_K M : \sigma$ ,  $M \rightarrow_\beta N$  и  $\vdash_K N : \tau$ , то необязательно, что  $\sigma = \tau$ .
- (b) Покажите, что в исчислении по Чёрчу «полное» свойство распространения типизации также не выполняется:

найдутся такие  $M, N, \sigma$ , что  $\vdash_C N : \sigma$ ,  $M \rightarrow_\beta N$ , но  $\nvdash_C M : \sigma$ .

Но при этом в исчислении по Чёрчу выполнено «ограниченное» свойство распространения типизации:

если  $\vdash_K M : \sigma$ ,  $M \rightarrow_\beta N$  и  $\vdash_K N : \tau$ , то тогда  $\sigma = \tau$ .

2. Покажите, что никакие связи в ИИВ не выражаются друг через друга: то есть, нет такой формулы  $\varphi(A, B)$  из языка интуиционистской логики, не использующей связку  $\star$ , что  $\vdash A \star B \rightarrow \varphi(A, B)$  и  $\vdash \varphi(A, B) \rightarrow A \star B$ . Покажите это для каждой связки в отдельности:

- (a) конъюнкция;
- (b) дизъюнкция;
- (c) импликация;
- (d) отрицание.

3. Рассмотрим алгоритм построения системы уравнений, а именно случай, когда рассматривается два разных вхождения одинакового по тексту применения. Например,  $(x \ x) (x \ x)$  имеет два разных вхождения одной и той же аппликации  $x \ x$ . Всегда ли для корректной работы алгоритма достаточно одной типовой переменной  $\beta_{xx}$  для этих двух вхождений, или нужны две разные  $\beta_{xx}^L$  и  $\beta_{xx}^R$ ? Примечание: при одной переменной для обеих аппликаций система в данном случае, очевидно, несовместна:  $\beta_{xx} \neq \beta_{xx} \rightarrow \sigma$ . Но контрпримером это не является, поскольку типа у данного выражения всё равно нет.

4. Предложим альтернативные аксиомы для конъюнкции:

$$\frac{\Gamma \vdash \alpha \quad \Gamma \vdash \beta}{\Gamma \vdash \alpha \ \& \ \beta} \text{ Введ. } \& \qquad \frac{\Gamma \vdash \alpha \ \& \ \beta \quad \Gamma, \alpha, \beta \vdash \gamma}{\Gamma \vdash \gamma} \text{ Удал. } \&$$

- (a) Предложите лямбда-выражения, соответствующие данным аксиомам; поясните, как данные выражения абстрагируют понятие «упорядоченной пары».
  - (b) Выразите изложенные в лекции аксиомы конъюнкции через приведённые в условии.
  - (c) Выразите приведённые в условии аксиомы конъюнкции через изложенные в лекции.
5. Постройте систему уравнений для  $Y$ -комбинатора и примените к ней алгоритм унификации (ожидается, что система окажется несовместной).

## Домашнее задание №3: сильная нормализуемость $\lambda_{\rightarrow}$ , система $F$

1. Найдите  $\llbracket \alpha \rightarrow \alpha \rrbracket$ .
2. Найдите  $\llbracket (\alpha \rightarrow \alpha) \rightarrow \alpha \rrbracket$ .
3. Покажите, что SN — насыщенное (постройте полноценное рассуждение по индукции для п.2 определения).
4. Покажите, что если  $\mathcal{A}$  и  $\mathcal{B}$  насыщены, то  $\mathcal{A} \rightarrow \mathcal{B}$  насыщенное.
5. Покажите, что построенная на лекции простая модель для ИИП второго порядка неполна.

6. Напомним, что мы можем задать  $\exists p.\varphi$  как  $\forall q.(\forall p.\varphi \rightarrow q) \rightarrow q$  (где  $q$  — некоторая свежая переменная). Покажите, что правила для квантора существования могут быть выведены из такого определения.
7. Требуется ли свобода для подстановки в правилах с квантором?

$$\frac{\Gamma \vdash \varphi[p := \theta]}{\Gamma \vdash \exists p.\varphi} \quad \frac{\Gamma \vdash \forall p.\phi}{\Gamma \vdash \phi[p := \theta]}$$

Если да — приведите пример доказуемой при отсутствии свободы для подстановки, но некорректной формулы. Если нет — предложите доказательство корректности правил при любых подстановках.

8. Пусть  $\Gamma \vdash \varphi$ . Всегда ли можно перестроить доказательство  $\varphi$ , добавив ещё одну гипотезу:  $\Gamma, \psi \vdash \varphi$ ? Если нет, каковы могли бы быть ограничения на  $\psi$ ?
9. Пусть  $\Gamma \vdash \forall x.\varphi$ . Верно ли тогда, что  $\Gamma \vdash \forall y.\varphi[x := y]$ ? Если это неверно в общем случае, возможно, это верно при каких-то ограничениях? В случае наличия ограничений приведите надлежащие контрпримеры.
10. Перенесите в систему  $F$  из бестипового лямбда-исчисления следующие функции — иными словами, постройте их обобщение для системы  $F$  (приведите обобщённое выражение, укажите его тип и докажите его). Например, можно рассмотреть  $I = \Lambda\pi.\lambda p^\pi.p \rightarrow p$ .
  - (a) S, K.
  - (b) инъекции и *case* (операции с алгебраическим типом);
  - (c) истина, ложь, исключающее или;
  - (d) черчёвский нумерал (он должен иметь тип  $\forall\alpha.(\alpha \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha)$ ) и инкремент;
  - (e) возведение в степень:  $\lambda m.\lambda n.n$   $m$ ;
  - (f) вычитание единицы (трюк зуба мудрости) и вычитание.

## Домашнее задание №4: экзистенциальные типы, типовая система Хиндли-Милнера

1. Постройте экзистенциальный тип для очереди, и реализуйте его с помощью двух стеков. Реализацию напишите на Хаскеле, используя `AbstractStack` с лекции как АТД стека (возможно, этот пример надо будет расширить нужными вам методами), и реализуйте какой-нибудь простой классический алгоритм с её помощью. Как, интересно, осуществить инстанциацию вложенного абстрактного типа данных? Придумайте.

Как с помощью двух стеков можно реализовать очередь со средним временем доступа  $\Theta(1)$ : входные значения кладём во входной стек, выходные достаём из выходного, при исчерпании — переносим всё из входного в выходной:

Входной стек	Выходной стек	Действие
$\square \rightarrow [1]$	$\square$	<i>push 1</i>
$[1] \rightarrow [3; 1]$	$\square$	<i>push 3</i>
$[3; 1] \rightarrow \square$	$\square \rightarrow [1; 3] \rightarrow [3]$	<i>pull</i>
$\square \rightarrow [5]$	$[3]$	<i>push 5</i>

2. Выразите дизъюнкцию через квантор существования в ИИП 2 порядка, а также алгебраический тип через экзистенциальный.
3. Покажите, что если  $rk(\sigma, 1)$ , то для выражения  $\sigma$  найдётся эквивалентное  $\sigma'$  с поверхностными кванторами.
4. Покажите, что в предыдущем задании также имеется изоморфизм типов: существует биективная функция  $\sigma \rightarrow \sigma'$ , которую можно выразить лямбда-выражениями.
5. Рассмотрим QuickSort:

```
let rec quick l = match l with
  [] -> []
  | l1 :: ls -> List.filter (fun x -> x < l1) ls @ [l1] @ List.filter (fun x -> x >= l1)
```

Укажите полные типовые схемы в системе НМ для всех функций, участвующих в данном примере (тип списка раскрывать не надо).

6. Заметим, что список — это «параметризованные» числа в аксиоматике Пеано. Число — это длина списка, а к каждому штриху мы присоединяем какое-то значение. Операции добавления и удаления элемента из списка — это операции прибавления и вычитания единицы к числу.

Рассмотрим тип «бинарного списка»:

```
type 'a bin_list = Nil | Zero of (('a*'a) bin_list) | One of 'a * (('a*'a) bin_list);;
```

Заметим, что здесь мы рассматриваем двоичную запись числа (чередующиеся `Zero` и `One`) — двоичную запись длины бинарного списка, и элемент двоичной записи номер  $n$  хранит  $2^n$  или  $2^n + 1$  значение (в зависимости от типа элемента). Например, 5-элементный список:

```
One ("a", Zero (One (((("b","c"),("d","e")), Nil)))
```

По идее, операция добавления элемента к списку записывается на языке Окамль вот так (сравните с прибавлением 1 к числу в двоичной системе счисления):

```
let rec add elem lst = match lst with
  Nil -> One (elem,Nil)
  | Zero tl -> One (elem,tl)
  | One (hd,tl) -> Zero (add (elem,hd) tl)
```

Однако, тип этой функции Окамль вывести автоматически не может, его надо указывать явно, чтобы код скомпилировался:

```
let rec add : 'a . 'a -> 'a bin_list -> 'a bin_list = fun elem lst -> match lst with
```

- Какой тип имеет `add` в (расширенной) системе  $F$  (напомним, поскольку функция рекурсивна, она должна использовать  $Y$ -комбинатор в своём определении)? Считайте, что семейство типов `bin_list 'a` предопределено и обозначается как  $\tau_\alpha$ . Также считайте, что определены функции `roll` и `unroll` с надлежащими типами. Какой ранг имеет тип этой функции? Почему этот тип не выразим в типовой системе Хиндли-Милнера?
  - Предложите функции для печати списка и для удаления элемента списка (головы).
  - Предложите функцию для эффективного соединения двух списков (источник для вдохновения — сложение двух чисел в столбик).
  - Предложите функцию для эффективного выделения  $n$ -го элемента из списка.
7. Рассмотрим следующий код на Окамле, содержащий определения чётрчевских нумералов и некоторых простых операций с ними:

```
let zero = fun f x -> x;;
let plus1 a = fun f -> fun x -> a f (f x);;
let power m n = n m;;

let two = plus1 (plus1 zero);;
let two2 = fun f x -> f (f x);;

let e = power two two;;          (* не компилируется *)
let e2 = power two2 two2;;       (* компилируется и работает *)
```

Разберите вывод типов в этом фрагменте (относительно типовой системы Хиндли-Милнера) и поясните, почему:

- определение `e2` компилируется и работает (предъявите доказательство типа в системе НМ);
- определение `e` не компилируется (например, примените алгоритм  $W$  и покажите шаг, где он выведет ошибку).

## Домашнее задание №5: Обобщённая система типов, гомотопическая теория типов, язык Аренд

- Задача на доказательство сильной нормализуемости  $\lambda_{\rightarrow}$ : найдите примеры лямбда-термов, не принадлежащие (1) множеству  $\llbracket \alpha \rightarrow \alpha \rrbracket$  и (2) множеству  $\llbracket (\alpha \rightarrow \alpha) \rightarrow \alpha \rrbracket$  (для выполнения задания надо выполнить оба пункта).
- Укажите тип (род) в исчислении конструкций для следующих выражений (при необходимости определите типы используемых базовых операций и конструкций самостоятельно):

- В алгебраическом типе `'a option = None | Some 'a` предложите тип (род) для: `Some`, `None` и `option`.
- Пусть задан род `nonzero : * → *`, выбрасывающий нулевой элемент из типа. Например, `nonzero unsigned` — тип положительных целых чисел. Тогда, для кода

```
template<typename T, T x>
struct NonZero { const static std::enable_if_t<x != T(0), T> value = x; };
```

предложите тип (род) поля `value`.

- Предложите выражение на языке C++ (возможно, использующее шаблоны), имеющее следующий род (тип):

- $* \rightarrow * \rightarrow *$ ;  $* \rightarrow \text{unsigned}$
- $\text{int} \rightarrow (* \rightarrow *)$
- $(* \rightarrow \text{int}) \rightarrow *$
- $\Pi x^*. \lambda n^{\text{int}}. F(n, x)$ , где

$$F(n, x) = \begin{cases} \text{int}, & n = 0 \\ x \rightarrow F(n - 1, x), & n > 0 \end{cases}$$

- Определите функции из следующих частей  $\lambda$ -куба (в обобщённой типовой системе) и докажите их тип:

- $(\square, *)$
- $(*, \square)$
- $(\square, \square)$

- Рассмотрим правый дальний нижний угол  $\lambda$ -куба  $(\{(*, *); (*, \square); (\square, *)\})$ . Можно предположить, что тогда в такой системе возможны и функции рода  $f : * \rightarrow *$  (как композиция функций  $p : * \rightarrow \alpha$  и  $q : \alpha \rightarrow *$  — например, можно кодировать тип его именем, затем по имени типа восстанавливать сам тип обратно). Почему всё-таки такие функции в обобщённых типовых системах невозможны без четвёртого элемента  $(\square, \square)$ ?

- Какова должна быть топология на множестве пар натуральных чисел (интуитивно мы будем понимать эти пары как рациональные числа, пары «числитель-знаменатель»), чтобы непрерывными были бы те и только те функции, для которых выполнено  $f(p, q) = f(p', q')$  для всех таких  $p, p', q$  и  $q'$ , что  $p \cdot q' = p' \cdot q$ . Напомним, что равенство мы понимаем как наличие непрерывного пути между точками.

- Докажите, приведя компилирующуюся программу на языке Аренд (возможно, вам потребуются функции и приёмы, изложенные в документации по языку: <https://arend-lang.github.io/documentation/tutorial/PartI/>):

- ассоциативность сложения;
- коммутативность сложения;
- коммутативность умножения;
- дистрибутивность:  $(a + b) \cdot c = a \cdot c + b \cdot c$ ;
- куб суммы:  $(a + 1)^3 = a^3 + 3 \cdot a^2 + 3 \cdot a + 1$ .

- Определим, что  $x$  делится на  $p$ , если обитаем тип `\Sigma (q : Nat) (p * q = x)`.

- Покажите, что если  $x$  делится на 6, то  $x$  делится и на 3;

- (b) Покажите, что  $x!$  делится на  $x$ ;
  - (c) Покажите, что если  $x$  делится на  $y$  и  $y$  делится на  $z$ , то  $x$  делится на  $z$ ;
9. Определите предикат (т.е. функцию с надлежащим типом) для формализации понятия простого числа `isPrime`. Покажите, что:
- (a) 3 и 11 — простые числа;
  - (b) Произведение простых чисел не просто;
  - (c) 2 — единственное чётное простое число.
10. Определим отношение «меньше» на натуральных числах так (с помощью индуктивного типа, обобщения алгебраического типа данных — зависимого типа, в котором при разных значениях аргументов типа допустимы разные конструкторы):

```
\data NatLessEq (a b : Nat) \with
| 0, m => natlesseq-zero
| suc m, suc n => natlesseq-next (NatLessEq m n)
```

Например, конструктор `natlesseq-zero` можно использовать только если первый аргумент типа — число 0. А конструктор `natlesseq-next` применим только если первый аргумент больше 0; при этом данный конструктор требует значение типа с определёнными аргументами в качестве своего аргумента.

Будем говорить, что  $a \preceq b$  тогда и только тогда, когда `NatLessEq a b` обитаем. Например, утверждение  $1 \preceq 3$  доказывается так:

```
\func one-le-three : NatLessEq 1 3 => natlesseq-next (natlesseq-zero)
```

В самом деле, `natlesseq-zero` может являться конструктором типа `NatLessEq 0 b`, а тогда

```
natlesseq-next (natlesseq-zero) : NatLessEq 1 (b+1)
```

Унифицировать  $b + 1$  и 3 компилятор (в данном случае) может самостоятельно, и потому код выше проходит проверку на корректность.

Докажите (везде предполагается, что  $a, b, c : \text{Nat}$ , если не указано иного):

- (a)  $a \preceq b$  тогда и только тогда, когда  $a$  меньше или равно  $b$  в смысле натуральных чисел (здесь требуется рассуждение на мета-языке).
  - (b)  $a \preceq a + b + 1$ ; то есть, определите функцию  

```
\func n-less-sum (a b : Nat) : NatLessEq a (a Nat.+ suc b)
```
  - (c) Если  $a \preceq b$ , то  $a + c \preceq b + c$
  - (d) Если  $a \preceq b$  и  $c \preceq d$ , то  $a \cdot c \preceq b \cdot d$
  - (e)  $a \preceq 2^a$
  - (f) Транзитивность: если  $a \preceq b$  и  $b \preceq c$ , то  $a \preceq c$
  - (g)  $a \preceq b \vee b \preceq a$
  - (h) Найдите стандартное определение отношения «меньше» в библиотеке Аренда (`Nat.<`) и докажите, что  $a \preceq b$  тогда и только тогда, когда  $a < b$  или  $a = b$  (реализуйте функции  

```
there (p : NatLessEq a b) : Data.Or (a Nat.< b) (a = b)
```

и обратную к ней).
  - (i) Покажите, что  $a \preceq b$  тогда и только тогда, когда  $\exists k^{\mathbb{N}_0}. a + k = b$ .
11. С точки зрения изоморфизма Карри-Ховарда индуктивные типы можно воспринимать как аналоги предикатов. В этом задании надо построить индуктивные типы для различных предикатов:
- (a) Факториал (`IsFact n`), который будет обитаем только для таких  $n$ , что  $n = k!$ . Докажите на языке Аренд, что `IsFact (1 · 2 · 3 · ... · n)` всегда обитаем, а тип `IsFact 3` — необитаем.
  - (b) Наибольший общий делитель двух чисел `GCD x a b`; *указание/пожелание*: воспользуйтесь алгоритмом Эвклида.
  - (c) Ограниченное натуральное число `IndFin n`, обитателями типа являются только те числа, которые меньше  $n$ . В стандартной библиотеке `Fin` определён через натуральные числа; сделайте это исключительно через индуктивные типы. Покажите, что если  $x : \text{IndFin } m$  и  $y : \text{IndFin } n$ , то  $x + y : \text{IndFin } (m + n)$ .

## Домашнее задание №6: Иерархии универсумов

1. Рассмотрим следующее доказательство уникальности элементов списка:

```
\func not-member (A : \Type) (elem : A) (l : List A) : \Type \elim l
| nil => \Sigma
| :: hd tl => \Sigma (Not (hd = elem)) (not-member A elem tl)

\func unique-list (A : \Type) (l : List A) : \Type \elim l
| nil => \Sigma
| :: hd tl => \Sigma (not-member A hd tl) (unique-list A tl)
```

Докажем, что список  $[0, 1, 2]$  состоит из уникальных элементов:

```
\func r-unique => unique-list Nat (0 :: 1 :: 2 :: nil)
\func x : r-unique => ((contradiction, (contradiction, ())), ((contradiction, ()), (((), ())))
```

- (a) Напишите функцию, строящую список  $b$  натуральных чисел от 0 до  $n$  и доказательство `unique-list Nat b`.
- (b) Покажите, что если  $a_0 < a_1 < \dots < a_n$ , то список  $[a_0, a_1, \dots, a_n]$  уникальный.
- (c) Покажите, что если  $n \geq 2$  и  $a_k \neq a_{k+1}$  при  $0 \leq k < n$ , то необязательно, что список  $[a_0, a_1, \dots, a_n]$  — уникальный.
2. Определите тип `Perm n` — его элементами должны быть те и только те списки чисел, которые являются перестановкой  $n$  элементов — и покажите, что:
- (a)  $0, 1, 2, \dots, n-1$  — перестановка  $n$  элементов;
- (b) определите, чему равна сумма всех элементов перестановки — и докажите что это действительно так для любого  $n$ ;
- (c) всего существует 6 элементов в типе `Perm 3` (то есть, существует такой список из 6 элементов, каждые два элемента которого не равны друг другу, и если  $x : \text{Perm } 3$ , то  $x$  — элемент данного списка).
3. Как вы помните из лекции, в языке Аренд существует иерархия вложенных типовых универсумов. Если в типе отсутствует упоминание `\Type`, то данный тип принадлежит универсуму 0. Однако, если в типе упоминается `\Type k`, то тип принадлежит универсумам, не меньшим  $k+1$ . Уровень универсума обозначается специальным ключевым словом `\lp`. Над индексами можно проводить простые операции и сопоставление с образцом (`\suc\lp`). Более подробно можно это прочесть в документации по языку Аренд:

<https://arend-lang.github.io/documentation/tutorial/PartI/universes.html>

Рассмотрим определения:

```
\func ChurchT (x : \Type) => (x -> x) -> (x -> x)
\func Church => \Pi (x : \Type) -> ChurchT x
\func Zero : Church => \lam t f x => x

\func incT (t : \Type) (n : ChurchT t) => \lam f x => n f (f x)
\func pair_plus (type : \Type) (pair : \Sigma (ChurchT type) (ChurchT type)) :
  \Sigma (ChurchT type) (ChurchT type) => (pair.2, incT type pair.2)
\func dec (n : Church) : Church => \lam (t : \Type) =>
  (n (\Sigma (ChurchT t) (ChurchT t)) (pair_plus t) (Zero t, Zero t)).1
\func sub (a : Church (\suc\lp)) (b : Church) => a (Church \lp) dec b
```

Определите, развивая определения выше:

- (a) Операцию умножения.
- (b) Операцию «деление на три» (естественно, в версии, не использующей  $Y$ -комбинатор).
- (c) Операцию возведения в степень, определяющуюся как  $\lambda m. \lambda n. n \ m$ .
- (d) Деление.
- (e) Вычисление факториала.



#### 4. Введём тип данных `IsEven`:

```
\data IsEven (n : Nat) \elim n
| 0 => zero-is-even
| (suc (suc k)) => next-next (IsEven k)
```

Доказать, что этот тип является утверждением, можно например так:

```
\func all-even-different (n : Nat) (a b : IsEven n) : a = b \elim n, a, b
| 0, zero-is-even, zero-is-even => idp
| suc (suc n), next-next a, next-next b => pmap next-next (all-even-different n a b)

\func is-even-isProp (n : Nat) : isProp (IsEven n) => all-even-different n
```

Обратите внимание: по переменным  $n$ ,  $a$  и  $b$  производится *элиминация*, то есть множество значений переменных разбивается на фрагменты (в соответствии с конструкторами типа данных), и доказательство утверждения проводится независимо для каждого фрагмента; в частности, цель доказательства изменяется и просходит замена переменных  $a$  и  $b$  на сопоставляемые варианты (вместо ожидаемого типа  $a = b$  мы ожидаем тип `zero-is-even = zero-is-even`, и т.п.). Сравните с лекцией про элиминаторы, каждый из вариантов — тело отдельной функции-элиминатора.

Чтобы воспроизвести тот же эффект в конструкции `\case`, нужно указывать ключевое слово `\elim` перед каждой элиминируемой переменной:

```
\case \elim n, \elim a, \elim b \with { ... }
```

Полный код, определяющий тип `IsEven` (вместе с доказательством того, что тип — утверждение), выглядит так:

```
\data IsEven (n : Nat) \elim n
| 0 => zero-is-even
| (suc (suc k)) => next-next (IsEven k)
\where {
  \func all-even-different ... -- скопируйте код функции сюда
  \use \level is-even-isProp (n : Nat) : isProp (IsEven n) => all-even-different n
}
```

Однако, незавершённым остаётся доказательство разрешимости типа `IsEven n`. Восполните лакуны:

```
\func even-is-dec (a : Nat) : Dec (IsEven a) \elim a
| 0 => yes zero-is-even
| 1 => no {?}
| suc (suc a) => {?}
```

5. Справедливости ради, в предыдущем задании компилятор сам может догадаться, что `IsEven` — утверждение. Но далеко не для всех типов это очевидно, и тогда функция с префиксом `\use \level` становится необходимой. Например, в следующем типе «простое число» данная функция должна доказать, что любые два значения типа при данном  $x$  равны:

```
\data Div3 (x : Nat)
| remainder-zero (Exists (p : Nat) (p Nat.* 3 = x))
| remainder-one (Exists (p : Nat) (p Nat.* 3 Nat.+ 1 = x))
| remainder-two (Exists (p : Nat) (p Nat.* 3 Nat.+ 2 = x))
\where \use \level levelProp {x : Nat} (a b : Div3 x) : a = b => {?}
```

- Замените `{?}` в тексте выше на корректное доказательство.
- Определите функцию, которая бы по  $x$  и значению  $\exists pq. 3 \cdot p + q = x \ \& \ 0 \leq q < 3$  возвращала бы `Div3 x` (понятно, можно разделить  $x$  на 3, но нам уже результат деления дали вторым аргументом — задача в том, чтобы им воспользоваться).
- Постройте аналогичный тип `Prime x` для простых чисел — с тремя вариантами `less-than-two`, `is-prime`, `is-composite` — и определите функцию, строящую по числу значение данного типа.

- (d) Покажите, что в типе `Prime (x*x + 2*x + 1)` всегда (кроме граничных случаев) обитает вариант `is-composite`.
- (e) Покажите, что в типе

```
\data SuperDec (P : \Prop)
| sure P
| nope (P -> Empty)
| neither ((P || (P -> Empty)) -> Empty)
\where \use \level superDecProp {P : \Prop} (a b : SuperDec P): a = b => {?}
```

вариант `neither` невозможен (также, заполните пропущенное доказательство `superDecProp`).

6. Рассмотрим определение целых чисел как упорядоченной пары:  $\langle a, b \rangle \subseteq \mathbb{N}^2$ , причём  $\langle a, b \rangle \approx \langle c, d \rangle$ , если  $a + d = b + c$ . Построим соответствующий тип данных,  $\mathbb{N}^2 / \approx$ :

```
\data Integer
| int_data (l r : Nat)
| int_eq (a b c d : Nat) (a Nat.+ d = b Nat.+ c) : int_data a b = int_data c d
```

Обратите внимание на второй конструктор `int_eq` — это специальный конструктор для отношения эквивалентности, он постулирует равенство между элементами, и его можно использовать для доказательства такого равенства. Указание на особую роль конструктора — указание его типа, и значением конструктора являются не сами элементы типа `Integer` (как это имеет место в случае `int_data`), а пути между элементами типа `Integer`. Покажем, например, что  $\langle 1, 3 \rangle = \langle 0, 2 \rangle$ :

```
\func r : int_data 1 3 = int_data 0 2 => int_eq 1 3 0 2 idp
```

Теперь мы можем определить функции на целых числах:

```
\func inc (a : Integer) : Integer \elim a
| int_data l r => int_data (suc l) r
| int_eq a b c d proof => int_eq (suc a) b (suc c) d (pmap suc proof)
```

Обратите внимание, мы должны указать не только образы для всех элементов `Integer` (первый случай), но и показать, что равные элементы перешли в равные (второй случай). А именно, нам потребовалось доказать, что операция прибавления 1 вернёт эквивалентные числа для эквивалентных аргументов: если  $\langle a, b \rangle \approx \langle c, d \rangle$ , то  $\langle a + 1, b \rangle \approx \langle c + 1, d \rangle$ .

- (a) Определите функцию `isPositive : Integer -> \Type`, результат которой обитает тогда и только тогда, когда аргумент — положительное число.
- (b) Покажите `Dec (isPositive k)` для любого целого  $k$ .
- (c) Определите функцию `dec : Integer -> Integer`, уменьшающую число на 1.
- (d) Докажите, что `inc (dec x) = x`.
- (e) Определите функцию `abs : Integer -> Nat`, возвращающую модуль целого числа.
- (f) Определите функцию `plus : Integer -> Integer -> Integer`, складывающую два числа.
7. Заметим, что получившийся тип `Integer` множеством (`\Set`) не будет. Чтобы такое выполнялось, необходимо показать равенство равенств элементов ( $\prod x^{\text{Integer}}. \prod y^{\text{Integer}}. \prod a^{x=y}. \prod b^{x=y}. a = b$ ). Это можно сделать (точнее, *постулировать*), например, с помощью конструкции `\truncate`:

```
\truncate \data Integer : \Set
| int_data (l r : Nat)
| int_eq (a b c d : Nat) (a Nat.+ d = b Nat.+ c) : int_data a b = int_data c d
```

И далее, чтобы показать равенство равенств, мы сможем воспользоваться библиотечной функцией `Path.inProp` (без указания `\truncate` данный код не скомпилируется):

```
\func integer_eq (x y : Integer) (p1 p2 : x = y) : p1 = p2 => Path.inProp p1 p2
```

- (a) Поясните (на метаязыке, с точки зрения топологии), почему неусечённый тип `Integer` — не `\Set`.
- (b) Определите функцию `isSet: \Type -> \Type`, результат которой обитает если исходный тип является множеством — в качестве источника вдохновения можно взять `isProp`. Докажите, что `Nat` — множество.
- (c) Использовать конструкцию `\truncate` необязательно — вместо неё можно указать надлежащий дополнительный конструктор равенства (теперь для путей) — и указать надлежащую функцию `\use \level`:

```
| eq_eq (a b : Integer) (p1 p2 : a = b) : p1 = p2
  \where { \use \level asSet ... }
```

Дополните описание типа данных так, чтобы тип данных `Integer` оказался множеством без специальной конструкции `\truncate` (убедитесь, что `Path.inProp p1 p2` теперь определён для путей на `Integer`, и поэтому аналог `integer_eq` скомпилируется). Также исправьте определение функции `inc` из прошлого задания.

8. Научимся раскрывать усечённый тип данных (при возможности это сделать):

- (a) По  $\exists p.p' = x$  постройте такой  $p$ , что  $p' = x$ :

```
\func safe-dec (x : Nat) (Exists (p : Nat) (suc p = x)) : \Sigma (p : Nat) (suc p = x)
```

*Указание:* второй параметр нужен для того, чтобы исключить вариант  $x = 0$ .

- (b) По `not-equals : (x > y || x < y)` при  $x, y : \text{Nat}$  (обратите внимание, здесь применяется усечённое «или») получите  $x \neq y$ .
- (c) По  $x : \text{Nat}$  и  $p : \text{Exists } (p : \text{Nat}) (p * p = x)$  найдите `\Sigma (p : Nat) (p * p = x)` (мы здесь должны существенно использовать единственность натурального корня числа).

9. В предыдущих заданиях мы строили фактор-множества вручную. То же можно сделать с помощью библиотечного типа данных `Quotient`.

- (a) Постройте тип множества рациональных положительных чисел `Rational` как фактор-множество пар  $\langle a, b \rangle$  и  $\langle c, d \rangle$  по отношению «равны как простые дроби»:  $\langle a, b \rangle \approx \langle c, d \rangle$ , если  $a \cdot d = b \cdot c$  ( $a, c \in \mathbb{N}_0, b, d \in \mathbb{N}$ ).
- (b) Определите арифметические операции (сложение, умножение).
- (c) Постройте функцию `to_rat (arg: Nat) : Rational` и `from_rat` (выполняющую округление вниз). Покажите, что `\Pi (x : Nat) -> x = from_rat (to_rat x)`.

## Домашнее задание №7: Алгебраическая топология

1. Докажите, что  $\mathbb{R}^2$  и  $\mathbb{R} \times [0, +\infty)$  гомотопически эквивалентны, но не гомеоморфны.
2. Докажите, что буквы  $\mathbb{O}$  и  $\mathbb{Q}$  гомотопически эквивалентны, но не гомеоморфны.
3. Покажите, что для любых двух мощностей  $\alpha$  и  $\beta$  найдутся два гомотопически эквивалентных пространства  $A, B$ :  $|A| = \alpha, |B| = \beta$ .
4. Покажите, что пространство  $X$  линейно связно тогда и только тогда, когда любые отображения  $\{0\} \rightarrow X$  гомотопны.
5. Подсчитайте  $\pi_1(S^2)$ .
6. Рассмотрим деревья с топологией «открыты множества, содержащие вершины со всеми своими потомками». Поясните, какие деревья будут в такой топологии гомотопически эквивалентны.
7. Докажите, что  $(\text{Bool} = \text{Bool}) = \text{Bool}$ .

## Домашнее задание №8: Аксиома выбора, теорема Диаконеску

1. Покажите (на метаязыке), что в ИИВ из закона исключённого третьего следует разрешимость (разумеется, не пользуясь разрешимостью ИИВ). А также поясните, что из разрешимости следует закон исключённого третьего.
2. Сформулируйте на языке Аренд аксиому выбора для  $\backslash\text{Set}$ -ов. Покажите, что она является теоремой.
3. Покажите, что равенство для  $\backslash\text{Set}$ -ов разрешимо.
4. Определите сетоиды в Аренде. Покажите, что целые числа образуют сетоид.
5. Докажите теорему Диаконеску на Аренде с помощью сетоидов.
6. Поясните (на метаязыке) утверждение: «аксиома выбора — это аксиома о перестановке кванторов».
7. С помощью аксиомы выбора на Аренде покажите, что любая сюръективная функция из факторизованного множества (файл стандартной библиотеки `Relation/Equivalence.ard`, тип данных `Quotient`) в факторизованное имеет частичную обратную.
8. Покажите, что если любая сюръективная функция из факторизованного множества (файл стандартной библиотеки `Relation/Equivalence.ard`, тип данных `Quotient`) в факторизованное имеет частичную обратную, то выполнена аксиома выбора.

## Домашнее задание №9: парадокс Жирара

1. Напомним, что для ординала  $\alpha$  за  $\sigma\alpha$  мы обозначили множество меньших ординалов, а за  $\tau X$  — порядковый тип упорядоченного множества  $\langle X, < \rangle$ . Покажите, что для любого ординала  $\alpha$  выполнено  $\tau\sigma\alpha = \alpha$ . Верно ли, что  $\alpha = \sigma\alpha$ ?
2. Пусть  $f : S \rightarrow T$ . Определим  $f_* : \wp S \rightarrow \wp T$  поэлементно:  $f_*(X) = \{f(x) \mid x \in X\}$ .
  - (a) Покажите, что универсум  $\mathcal{U}$  парадоксален тогда и только тогда, когда  $\langle \wp\mathcal{U}, \sigma_*, \tau_* \rangle$  парадоксален.
  - (b) Рассмотрим  $\mathcal{U} = \{\langle A, \prec, a \rangle \mid (\prec) \subseteq A^2, a \in A\}$ . Пусть  $\sigma\langle A, \prec, a \rangle = \{\langle A, \prec, b \rangle \mid b \prec a\}$ . Тогда  $\sigma : \mathcal{U} \rightarrow \wp\mathcal{U}$ .  
Тогда мы можем ввести отношение на  $\wp\mathcal{U}$ . Будем считать  $Y < X$ , если  $Y \in \sigma_* X$ .  
Определим  $\tau X := \langle \wp\mathcal{U}, (<), X \rangle$ .  
Покажите, что  $\langle \mathcal{U}, \sigma, \tau \rangle$  — парадоксальный универсум.
3. Покажите, что  $\Omega \in \mathcal{U}$ .
4. Покажите невозможность фундированности  $\Omega$ . А именно, обоснуйте переходы и восполните пробелы в следующих рассуждениях:
  - (a) Лемма. Множество  $\{y \mid \neg\tau\sigma y < y\}$  индуктивно:
    - Рассмотрим  $x$ , что для всех  $y < x$  выполнено  $\neg\tau\sigma y < y$ .
    - Предположим, что  $\tau\sigma x < x$ .
    - Тогда  $\neg\tau\sigma\tau\sigma x < \tau\sigma x$ .
    - Но  $\tau\sigma\tau\sigma x = \tau\sigma u$  для некоторого  $u < x$ .
  - (b) Основная теорема:  $\Omega$  не фундировано.
    - Предположим, что  $\Omega$  фундировано.
    - $\tau\sigma\Omega$  имеет вид  $\tau\sigma\omega$  для некоторого фундированного  $\omega$ .
    - $\tau\sigma\Omega$  есть предшественник  $\Omega$ .
    - Однако,  $\neg\tau\sigma\Omega < \Omega$ , поскольку  $\Omega$  фундировано и  $\{y \mid \neg\tau\sigma y < y\}$  индуктивно.
5. Вспомним определение с лекции:
  - $\mathcal{U} = \Pi X : \square.((\wp X \rightarrow X) \rightarrow \wp X)$
  - $\tau = \lambda X : \wp\mathcal{U}. \lambda A : \square. \lambda c : (\wp A \rightarrow A). \lambda a : A. \varphi$ , где  $\varphi = \Pi P^{\wp A}. \Pi x^{\mathcal{U}}. (X \ x \rightarrow (P(c((x \ A) \ c)))) \rightarrow P \ a$
  - $\sigma = \lambda x : \mathcal{U}. ((x \ \mathcal{U}) \ \tau)$
  - (a) Найдите типы  $\tau$  и  $\sigma$ .

- (b) Покажите, что универсум  $(\mathcal{U}, \sigma, \tau)$  парадоксален, показав бета-эквивалентность: для всех  $X : \wp \mathcal{U}$  выполнено  $\sigma \tau X =_{\beta} \bigcap \{P \subseteq \mathcal{U} \mid \forall x \in X. \tau \sigma x \in P\}$ . В лямбда-исчислении нет примитива для аксиомы выделения, поэтому записать прямо формулу вида  $T = \{\tau \sigma x \mid x \in X\}$  не получится — но тот же результат можно получить, построив пересечение всех подмножеств  $\mathcal{U}$ , содержащих  $T$  (или, говоря ближе к языку лямбда-исчисления, построив конъюнкцию всех предикатов  $P$  со свойством  $X x \rightarrow P(\tau \sigma x)$ ).

## Домашнее задание №10: линейная логика, уникальные типы

- Как известно, интуиционистские связки могут быть выражены в линейном исчислении. Покажите соответствующие интуиционистские аксиомы для следующих способов выразить интуиционистские связки через линейные:
  - Интуиционистская дизъюнкция:  $A + B := !A \oplus !B$
  - Интуиционистская конъюнкция:  $A \times B := A \& B$
  - Альтернативная конъюнкция:  $A \times B := !A \otimes !B$
  - Покажите, что альтернативная конъюнкция влечёт обычную, но не наоборот:  $\langle !A \otimes !B \rangle \vdash A \& B$ , но  $\langle A \& B \rangle \not\vdash !A \otimes !B$ .
- Покажите следующие линейные тождества:
  - $\vdash \phi \multimap \phi$
  - $\vdash !\phi \multimap !!\phi$
  - $!(\alpha \& \beta) \equiv !\alpha \otimes !\beta$  (надо построить два доказательства,  $\varphi \vdash \psi$  и  $\psi \vdash \varphi$ )
- Дистрибутивность
  - Выполняется ли дистрибутивность для  $(\&)$  и  $(\oplus)$ ?
  - Выполняется ли дистрибутивность для  $(\&)$  и  $(\otimes)$ ?
- В языке Раст есть тип функции, которую можно вызвать только один раз: `FnOnce`. Утверждается, что функция `const x y = x` в типовой системе уникальных типов (как во второй части лекции) будет иметь тип  $t^u \rightarrow^{\times} s^v \rightarrow^u t^u$  (то есть, вернёт однократную функцию, если её первый аргумент уникальный). Покажите это, пользуясь правилами вывода с лекции. Каков будет тип этой функции в языке Раст (вопрос на дополнительный балл)?

## Дополнительные задания на баллы

- (15 баллов) Перенесите доказательство парадокса Жирара с языка Coq на Аренд: <https://coq.inria.fr/doc/master/stdlib/Coq.Logic.Hurkens.html>
- (15 баллов) Формализуйте и докажите малую теорему Ферма: если  $a$  ( $a \in \mathbb{N}$ ) не делится на простое число  $p$ , то

$$a^{p-1} \equiv 1 \pmod{p}$$