

ТЕОРЕТИЧЕСКИЕ ДОМАШНИЕ ЗАДАНИЯ

Теория типов, ИТМО, М3235-М3239, осень 2022 года

Домашнее задание №1: «вводная лекция»

1. Напомним определения с лекций:

Обозначение	лямбда-терм	название
T	$\lambda a. \lambda b. a$	истина
F	$\lambda a. \lambda b. b$	ложь
Not	$\lambda x. x \ F \ T$	отрицание
And	$\lambda x. \lambda y. x \ y \ F$	конъюнкция

Постройте лямбда-выражения для следующих булевских выражений:

- (a) Штрих Шеффера («и-не»)
- (b) Стрелка Пирса («или-не»)
- (c) Мажоритарный элемент от трёх аргументов (результат «истина», если истинны не менее двух аргументов)

2. Напомним определения с лекций:

$$f^{(n)} X ::= \begin{cases} X, & n = 0 \\ f^{(n-1)} (f X), & n > 0 \end{cases}$$

Обозначение	лямбда-терм	название
\bar{n}	$\lambda f. \lambda x. f^{(n)} x$	чёрчевский нумерал
$(+1)$	$\lambda n. \lambda f. \lambda x. n \ f \ (f \ x)$	прибавление 1
$IsZero$	$\lambda n. n \ (\lambda x. F) \ T$	проверка на 0

Обозначение	лямбда-терм	название
$MkPair$	$\lambda a. \lambda b. (\lambda x. x \ a \ b)$	создание пары
PrL	$\lambda p. p \ T$	левая проекция
PrR	$\lambda p. p \ F$	правая проекция
$Case$	$\lambda l. \lambda r. \lambda c. c \ l \ r$	case для алгебраического типа
InL	$\lambda l. (\lambda x. \lambda y. x \ l)$	левая инъекция
InR	$\lambda r. (\lambda x. \lambda y. y \ r)$	правая инъекция

Используя данные определения, постройте выражения для следующих операций над числами:

- (a) Вычитание 1
- (b) Вычитание
- (c) Деление
- (d) Сравнение двух чисел ($IsLess$) — истина, если первый аргумент меньше второго (могут потребоваться пары и/или вычитания)
- (e) Делимость

3. Проредуцируйте выражение и найдите его нормальную форму:

- (a) $\bar{2} \ \bar{2}$
- (b) $\bar{2} \ \bar{2} \ \bar{2}$
- (c) $\bar{2} \ \bar{2} \ \bar{2} \ \bar{2} \ \bar{2} \ \bar{2} \ \bar{2}$

4. Напомним определение Y-комбинатора: $\lambda f. (\lambda x. f \ (x \ x)) \ (\lambda x. f \ (x \ x))$.

- (a) Покажите, что выражение $Y \ f$ не имеет нормальной формы;
- (b) Покажите, что выражение $Y \ (\lambda f. \bar{0})$ имеет нормальную форму.
- (c) Покажите, что выражение $Y \ (\lambda f. \lambda x. (IsZero \ x) \ \bar{0} \ (f \ Minus1 \ x)) \ 2$ имеет нормальную форму.
- (d) Какова нормальная форма выражения $Y \ (\lambda f. \lambda x. (IsZero \ x) \ \bar{0} \ ((+1) \ (f \ Minus1 \ x))) \ \bar{n}$?

- (e) Какова нормальная форма выражения $Y (\lambda f. \lambda x. (IsZero\ x) \bar{1} (Mul2\ (f\ Minus1\ x))) \bar{n}$?
- (f) Определите с помощью Y -комбинатора функцию для вычисления n -го числа Фибоначчи.
5. Определим на языке Хаскель следующую функцию: `show_church n = show (n (+1) 0)` Убедитесь, что `show_church (\f -> \x -> f (f x))` вернёт 2. Пользуясь данным определением и его идеей, реализуйте следующие функции:
- (a) `int_to_church` — возвращает чёрчевский нумерал (т.е. функцию от двух аргументов) по целому числу. Каков точный тип результата этой функции?
- (b) сложение двух чёрчевских нумералов.
- (c) умножение двух чёрчевских нумералов.
- (d) можно ли определить вычитание 1 и вычитание? Что получается, а что — нет?
6. На лекции было использовано понятие свободы для подстановки.
- (a) Найдите лямбда-выражение, которое при однократной редукции требует переименования связанных переменных (редукция невозможна без переименования).
- (b) Заметим, что даже если мы запретим использовать одни и те же переменные в разных лямбда-абстракциях, это не будет решением проблемы переименований. Предложите лямбда-выражение, в котором (a) все лямбда-абстракции указаны по разным переменным; но (б) через некоторое количество редукций потребуются переименование связанных переменных.
7. Дадим определение: комбинатор — лямбда-выражение без свободных переменных.
- Также напомним определение:

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

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 .

Выразите в базисе SK :

- (a) $F = \lambda x. \lambda y. y$
- (b) $\bar{1}$
- (c) Not
- (d) Xor
- (e) InL
8. Чёрчевские нумералы соответствуют натуральным числам в аксиоматике Пеано.
- (a) Предложите «двоичные нумералы» — способ кодирования чисел, аналогичный двоичной системе (такой, при котором длина записи числа соответствует логарифму числового значения).
- (b) Предложите реализацию функции $(+1)$ в данном представлении.
- (c) Предложите реализацию лямбда-выражения преобразования числа из двоичного нумерала в чёрчевский.
- (d) Предложите реализацию функции сложения в данном представлении.
- (e) Предложите реализацию функции вычитания в данном представлении.
- (f) Какова вычислительная сложность арифметических операций с двоичными нумералами?

Домашнее задание №2: ещё о бестиповом лямбда-исчислении

1. Бесконечное количество комбинаторов неподвижной точки. Дадим следующие определения

$$\begin{aligned} L &:= \lambda abcdefghijklmnopqrstuvwxyzr. r(this\ is\ a\ fixed\ point\ combinator) \\ R &:= LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL \end{aligned}$$

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. 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 :

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

и надо выбрать самый левый из них:

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

- (a) Проведите аппликативную редукцию выражения 2 2.
- (b) Постройте выражение, использующее Y -комбинатор для вычисления факториала. Возможно ли его аппликативное вычисление, или оно не сможет завершиться?
- (c) Найдите лямбда-выражение, которое редуцируется медленнее при нормальном порядке редукции, чем при аппликативном, даже при наличии мемоизации.
3. 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.

Верно ли, что «нормальность» формы выражения может в процессе редукции только усиливаться (никакая — слабая заголовочная Н.Ф. — заголовочная Н.Ф. — нормальная форма)?

4. Заметим, что список в лямбда-выражении можно закодировать с помощью алгебраических типов. Напишите лямбда-выражение для:
- (a) вычисления длины списка;
- (b) функции *map* (построение нового списка из результатов применения функции к элементам старого);
- (c) суммы списка целых чисел.
5. Базис SKI не единственный. Рассмотрим базис $BCKW$:

$$B = \lambda x.\lambda y.\lambda z.x (y z)$$

$$C = \lambda x.\lambda y.\lambda z.x z y$$

$$K = \lambda x.\lambda y.x$$

$$W = \lambda x.\lambda y.x y y$$

- (a) Покажите, что базис $BCKW$ позволяет выразить любое выражение из базиса SKI .
- (b) Покажите, что любое выражение из базиса $BCKW$ может быть выражено в базисе SKI .

Домашнее задание №3: просто-типизированное лямбда-исчисление

1. Пусть фиксирован тип чёрчевского нумерала $\eta = (\alpha \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha)$. Найдите тип для следующих конструкций и постройте доказательство:
- (a) $\bar{2}$ (покажите, что его тип — η).
- (b) $(+1)$.
- (c) *Plus*.
- (d) *Mul* (не каждая реализация умножения имеет тип в просто-типизированном лямбда исчислении; вам требуется найти нужную)
2. Имеет ли тип — и какой:
- (a) операция вычитания 1 (выраженная через «трюк зуба мудрости»)? Общий ответ не требуется, достаточно рассмотреть какую-то одну реализацию.

- (b) операция вычитания $(\lambda m. \lambda n. m \ (-1) \ n)$?
 - (c) операция возведения в степень $(Power ::= \lambda m. \lambda n. n \ m)$?
 - (d) функция $\lambda x. Power \ x \ x$?
3. Каков тип:
- (a) комбинаторов S и K ;
 - (b) истины и лжи.
4. Рассмотрим полную интуиционистскую логику, с конъюнкцией, дизъюнкцией и ложью. Какой тип у следующих конструкций, и какие правила вывода интуиционистской логики им соответствуют (ответ требует демонстрации корректности этих правил для данных конструкций — то есть вывод про тип результата применения правила должен всегда иметь место для выражений соответствующего вида):
- (a) Упорядоченная пара (MkPair, PrL, PrR).
 - (b) Алгебраический тип (InL: $\lambda x. \lambda a. \lambda b. a \ x$, InR: $\lambda x. \lambda a. \lambda b. b \ x$, Case).
5. Докажите лемму о редукции (subject reduction lemma): если $A \rightarrow_\beta B$ и $\vdash A : \tau$, то $\vdash B : \tau$.
Верно ли обратное: если $A \rightarrow_\beta B$ и $\vdash B : \tau$, то $\vdash A : \tau$?
6. Как мы уже разбирали, $\nvdash x \ x : \tau$ в силу дополнительных ограничений правила

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

Найдите лямбда-выражение N , что $\nvdash N : \tau$ в силу ограничений правила

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

7. Верно ли, что $S = B(BW)(BBC)$? Если нет, то как правильно?

Домашнее задание №4: «Изоморфизм Карри-Ховарда»

1. Предложим альтернативные аксиомы для конъюнкции:

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

- (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.
2. *Вполне упорядоченным* множеством назовём такое линейно-упорядоченное отношение (\prec) множество S (и такой порядок назовём *полным*), что какое бы ни было множество $U \subseteq S$, в U найдётся наименьший элемент.
- (a) Покажите, что неотрицательные вещественные числа $[0, +\infty)$ — не вполне упорядоченное множество. Существуют ли конечные и счётные не вполне упорядоченные множества?
 - (b) Определим лексикографический порядок на \mathbb{N}^n : положим, что $\langle a_1, a_2, \dots, a_n \rangle \prec \langle b_1, b_2, \dots, b_n \rangle$, если найдётся такой k , что $a_1 = b_1, \dots, a_{k-1} = b_{k-1}$, но $a_k < b_k$. Покажите, что такой порядок — полный.
 - (c) Пусть S вполне упорядочено отношением (\prec) , определим $a \succ b := b \prec a$. Пусть $a_1 \succ a_2 \succ a_3 \succ \dots$ — строго монотонно убывающая последовательность значений из S . Покажите, что данная последовательность всегда имеет конечную длину.

3. Поясним название «алгебраические типы» — это семейство составных типов, позволяющих строить «алгебраические» выражения на типах:

название	обозначение	алгебраический смысл
тип-сумма, «алгебраический»	$\alpha \vee \beta$	$\alpha + \beta$
тип-произведение, пара	$\alpha \& \beta$	$\alpha \times \beta$
тип-степень, функция	$\alpha \rightarrow \beta$	β^α

Название «алгебраический» закрепилось в первую очередь за типом-суммой (видимо потому, что остальные типы имеют устоявшиеся названия), однако, может быть отнесено и к другим типам.

Поясните «типовый» (программистский) смысл следующих алгебраических тождеств — и постройте программы на Хаскеле, их доказывающие:

- (a) $\gamma \times (\alpha + \beta) = \gamma \times \alpha + \gamma \times \beta$.
 (b) $\gamma^{\alpha \times \beta} = (\gamma^\alpha)^\beta$. Как называется данное тождество?
 (c) $\gamma^{\alpha + \beta} = \gamma^\alpha \times \gamma^\beta$.

4. Напомним, что $\neg \alpha \equiv \alpha \rightarrow \perp$. Найдите лямбда-выражения, доказывающие:

- (a) Формулу де-Моргана $\neg(\alpha \vee \beta) \rightarrow \neg \alpha \& \neg \beta$.
 (b) Контрапозицию $(\alpha \rightarrow \beta) \rightarrow (\neg \beta \rightarrow \neg \alpha)$.
 (c) Закон исключённого третьего после применения теоремы Гливленко $\neg \neg(\alpha \vee \neg \alpha)$.

5. Какие аксиомы соответствуют базису *BCKW*? Покажите, что аксиома, соответствующая *S*, доказывается в этой аксиоматике.
 6. Выразите в Хаскеле *Y*-комбинатор. Каков его тип?
 7. Покажите, что типовая система Хаскеля противоречива.

Домашнее задание №5: «Реконструкция типа лямбда-выражений в просто-типизированном исчислении»

1. На лекции вводилась метрика для доказательства завершаемости алгоритма унификации: упорядоченная тройка $\langle x, y, z \rangle$, где x — количество уравнений в разрешённой форме (уравнений вида $a = \theta$, причём a входит в систему ровно один раз), z — количество уравнений типа $a = a$ и $\theta = b$. Смысл же параметра y не был раскрыт.

Каким взять параметр для y , чтобы получившаяся метрика строго монотонно убывала при каждом применении правил унификации?

2. Применив алгоритм, рассказанный на лекции, найдите тип для комбинатора *S*.
 3. Применив алгоритм, рассказанный на лекции, покажите отсутствие типа у *Y*-комбинатора.
 4. Исчислением предикатов второго порядка назовём исчисление со следующим языком:

$$\Phi ::= x | (\Phi \rightarrow \Phi) | (\forall x. \Phi)$$

Содержательное отличие от исчисления высказываний — наличие квантора всеобщности и правил вывода для его введения и удаления:

$$\frac{\Gamma \vdash \phi}{\Gamma \vdash \forall p. \phi} (p \notin FV(\Gamma)) \quad \frac{\Gamma \vdash \forall p. \phi}{\Gamma \vdash \phi[p := \Theta]}$$

Докажите, что следующие связки могут быть выражены в таком исчислении (то есть, покажите, что соответствующие формулы удовлетворяют соответствующим правилам вывода для интуиционистского исчисления высказываний):

- (a) $\psi \& \varphi := \forall g. (\psi \rightarrow \varphi \rightarrow g) \rightarrow g$
 (b) $\psi \vee \varphi := \forall g. (\psi \rightarrow g) \rightarrow (\varphi \rightarrow g) \rightarrow g$
 (c) $\perp := \forall a. a$
 (d) $\exists a. \psi := \forall g. (\forall a. (\psi \rightarrow g)) \rightarrow g$

Для квантора существования правила вывода следующие:

$$\frac{\Gamma \vdash \phi[p := \psi]}{\Gamma \vdash \exists p. \phi} \quad \frac{\Gamma \vdash \exists p. \phi \quad \Gamma, \varphi \vdash \psi}{\Gamma \vdash \psi} (p \notin FV(\Gamma, \psi))$$

Домашнее задание №6: «Логика второго порядка и система F»

1. Требуется ли свобода для подстановки в правилах с квантором?

$$\frac{\Gamma \vdash \phi}{\Gamma \vdash \forall p. \phi} (p \notin FV(\Gamma)) \quad \frac{\Gamma \vdash \forall p. \phi}{\Gamma \vdash \phi[p := \theta]}$$

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

2. Пусть $\Gamma \vdash \varphi$. Всегда ли можно перестроить доказательство φ , добавив ещё одну гипотезу: $\Gamma, \psi \vdash \varphi$? Если нет, каковы могли бы быть ограничения на ψ ?
3. 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.
4. Перенесите в систему F из бестипового лямбда-исчисления следующие функции (приведите выражение, укажите его тип и докажите его):
 - (a) инъекции и *case* (операции с алгебраическим типом);
 - (b) истина, ложь, исключающее или;
 - (c) черчёвский нумерал (он должен иметь тип $\forall \alpha. (\alpha \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha)$) и инкремент;
 - (d) возведение в степень: $\lambda m. \lambda n. n \ m$;
 - (e) вычитание единицы (трюк зуба мудрости) и вычитание.
5. Напомним определения с лекции:

$$\frac{\Gamma \vdash M : \varphi[\alpha := \theta]}{\Gamma \vdash (\text{pack } M, \theta \text{ to } \exists \alpha. \varphi) : \exists \alpha. \varphi} \quad \frac{\Gamma \vdash M : \exists \alpha. \varphi \quad \Gamma, x : \varphi \vdash N : \psi}{\Gamma \vdash \text{abstype } \alpha \text{ with } x : \varphi \text{ in } M \text{ is } N : \psi} \quad \alpha \notin FV(\Gamma, \psi)$$

Покажите, что *pack* и *abstype* могут быть заданы так:

$$\text{pack } M, \theta \text{ to } \exists \alpha. \varphi = \Lambda \beta. \lambda x^{\forall \alpha. \varphi \rightarrow \beta}. x \theta M$$

$$\text{abstype } \alpha \text{ with } x : \varphi \text{ in } M \text{ is } N : \psi = M \psi (\Lambda \alpha. \lambda x^{\varphi}. N)$$

То есть, соответствующие правила вывода будут выполнены для так заданных выражений.

6. У правил вывода с кванторами для системы F есть ограничения. Покажите, что эти ограничения существенны: что без них с помощью кванторов можно типизировать лямбда-выражения, разрушающие систему F (то есть, нарушающие какие-то её существенные свойства, например, делающие её противоречивой).
7. (a) Разработайте интерфейс и реализацию для абстрактного типа данных «множество» (функции создания пустого множества, добавления, удаления, проверки наличия элемента в множестве). Напишите тестовую программу, использующую данный АТД.
(b) Сделайте по АТД «множество» соответствующий экзистенциальный тип в системе F , перенесите его в Хаскель и дайте реализацию этого типа. Приспособьте тест из предыдущего пункта.
8. Переформулируйте систему F в исчислении по Карри: укажите новые схемы аксиом для кванторов всеобщности и существования.
9. Переформулируйте операции *abstype* и *pack* для исчисления по Карри, укажите соответствующие им лямбда-выражения и покажите, что эти выражения соответствуют аксиомам.

Домашнее задание №7: «Типовая система Хиндли-Милнера»

1. Приведите правило вывода (обозначавшееся на лекции как $7'$), типизирующее `let` для рекурсивной функции:

```
let rec x = A in B
```

Покажите, что это правило делает систему противоречивой.

2. Покажите, что если два логических выражения в логике второго порядка эквивалентны ($\vdash \varphi \rightarrow \psi$ и $\vdash \psi \rightarrow \varphi$), то соответствующие типы либо одновременно обитаемы, либо одновременно необитаемы.
3. Покажите, как по значению типа $\forall \alpha. \beta \rightarrow \varphi(\alpha)$ строить значение типа $\beta \rightarrow \forall \alpha. \varphi(\alpha)$ и наоборот $\alpha \notin FV(\beta)$.
4. Типовая система Хиндли-Милнера типизирована по Чёрчу. Измените правила и язык так, чтобы она стала типизирована по Карри.
5. *О выразительной силе HM.* 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);;
```

Если бы такое можно было выразить в типовой системе Хиндли-Милнера, то операция добавления элемента к списку записалась бы 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)
```

- (a) Какой тип имеет `add` (обратите внимание на ключевое слово `rec`: для точного указания соответствующего лямбда-выражения и вывода типа необходимо использовать Y -комбинатор)? Считайте, что семейство типов `bin_list 'a` предопределено, и обозначается как τ_α . Также считайте, что определены функции `roll` и `unroll` с надлежащими типами.
 - (b) Какой ранг имеет тип этой функции? Почему этот тип не выразим в типовой системе Хиндли-Милнера?
 - (c) Предложите функцию для удаления элемента списка (головы).
 - (d) Предложите функцию для эффективного соединения двух списков (источник для вдохновения — сложение двух чисел в столбик).
 - (e) Предложите функцию для эффективного выделения n -го элемента из списка.
6. Рассмотрим следующий код на Окамле, содержащий определения чёрчевских 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 *)
```

Поясните, почему:

- (a) определение $e2$ компилируется и работает;
- (b) определение e не компилируется.

Пояснение должно содержать необходимые фрагменты вывода типа в системе Хиндли-Милнера, или должно показывать, что нужного вывода типа не существует.

7. Какой ранг имеют экзистенциальный тип и тип монады **ST** из Хаскеля?

Домашнее задание №8-9: «Обобщённая типовая система; язык Аренд»

1. Укажите тип (род) в исчислении конструкций для следующих выражений (при необходимости определите типы используемых базовых операций и конструкций самостоятельно):

- (a) В алгебраическом типе `'a option = None | Some 'a` предложите тип (род) для: `Some`, `None` и `option`.
- (b) 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`.

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

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

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

3. Определите функции из следующих частей λ -куба (в обобщённой типовой системе) и докажите их тип:

- (a) $(\square, *)$
- (b) $(*, \square)$
- (c) (\square, \square)

4. Рассмотрим правый дальний нижний угол λ -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) ?

5. Как отмечалось на занятии, мы рассматриваем множество натуральных чисел как множество с дискретной топологией (для того, чтобы функции из Nat в Nat были бы непрерывны). Поясните, почему дискретная топология гарантирует непрерывность любой такой функции? Напомним, что непрерывная функция — такая, у которой любой прообраз открытого множества открыт.

6. 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.

7. 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/>):

- (a) ассоциативность сложения;
- (b) коммутативность сложения;

- (c) коммутативность умножения;
 - (d) дистрибутивность: $(a + b) \cdot c = a \cdot c + b \cdot c$;
 - (e) куб суммы: $(a + 1)^3 = a^3 + 3 \cdot a^2 + 3 \cdot a + 1$.
8. Определим, что x делится на p , если обитает тип $\Sigma (q : \mathbb{N}) (p * q = x)$.
- (a) 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.

Домашнее задание №10: «Ещё доказательства»

1. Покажите, что если на множестве пар $Q = \mathbb{N}_0 \times \mathbb{N}$ задана топология с базой

$$B = \{S \subseteq Q \mid \forall \langle p, q \rangle. \forall \langle p', q' \rangle. p \cdot q' = p' \cdot q\}$$

то все функции $f : Q \rightarrow X$, для которых выполнено $f(\langle p, q \rangle) \sim f(\langle p', q' \rangle)$ для всех таких p, p', q и q' , что $p \cdot q' = p' \cdot q$ — непрерывны (волнистую стрелку мы понимаем как наличие непрерывного пути в пространстве X).

2. 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` обитает. 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 $a, b, c : \mathbb{N}$, 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 $\exists k^{\mathbb{N}_0}. a + k = b$.
3. 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 обитает только для таких n , что $n = k!$. Докажите на языке Аренд, что `IsFact (1 · 2 · 3 · ... · n)` всегда обитает, а тип `IsFact 3` — необитает.
 - (b) Наибольший общий делитель двух чисел `GCD x a b`; *указание/пожелание*: воспользуйтесь алгоритмом Евклида.
 - (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 $x + y : \text{IndFin } (m + n)$.
4. 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).

Домашнее задание №11: «Элиминаторы; предикативные уровни»

1. Чтобы было проще решать задачи из предыдущего задания, давайте решим дополнительную задачу на похожую тему. 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. 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.
3. Определите следующие свойства как пропы и покажите, что это так (докажите `isProp` для каждого, см. стандартную библиотеку):
- (a) Натуральное число простое.
 - (b) Натуральное число не делится на 3.
 - (c) x и y натуральные, $x < y$.
 - (d) Натуральное число — полный квадрат.
 - (e) Два натуральных числа имеют «нетривиальный» (не равный 1 или одному из этих чисел) общий делитель.

Домашнее задание №12: «Гомотопические уровни»

1. Напомним тип данных `IsEven` из лекции:

```

\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) => {?}

```

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

- 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 обитаем 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.
4. Заметим, что получившийся тип `Integer` множеством (`\Set`) не будет. Чтобы такое выполнялось, необходимо показать равенство равенств элементов ($\Pi x^{\text{Integer}}. \Pi y^{\text{Integer}}. \Pi a^{x=y}. \Pi b^{x=y}. a = b$). 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 множество. *Указание:* заметим, что путь $\langle 0, 0 \rangle = \langle 0, 0 \rangle$ можно получить композицией путей $\langle 0, 0 \rangle = \langle 1, 1 \rangle$ и $\langle 1, 1 \rangle = \langle 0, 0 \rangle$.
- (b) Определите функцию `isSet : \Type -> \Type`, результат которой обитаем 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.

5. Рассмотрим аксиому для квантора существования из исчисления предикатов (в гильбертовской форме):

```
\func exists-axiom {T : \Type} {A : T -> \Prop} (evidence : T) (proof : A evidence)
: Exists (x : T) (A x) => inP (evidence, proof)
```

Сформулируйте и докажите:

- (a) правило вывода из исчисления предикатов;
- (b) правило удаления квантора существования из исчисления предикатов в натуральном выводе:

$$\frac{\vdash \exists x. \varphi(x) \quad \varphi(\theta) \vdash \psi}{\vdash \psi}$$

6. 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 $\Sigma (p : \text{Nat}) (p * p = x)$ (мы здесь должны существенно использовать единственность натурального корня числа).

7. 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 $\forall x : \text{Nat} \rightarrow x = \text{from_rat } (\text{to_rat } x)$.