Теоретические домашние задания

Теория типов, ИТМО, M3232-M3239, весна 2025 года

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

1. Бесконечное количество комбинаторов неподвижной точки. Дадим следующие определения

$$\begin{array}{l} L := \lambda abcdefghijklmnopqstuvwxyzr.r(thisisafixedpointcombinator) \\ R := LLLLLLLLLLLLLLLLLLLLLLLLLL \end{array}$$

   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 32 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 τ называется необитаемым, если ни для какого P не выполнено $\vdash P : \tau$).

3. 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 σ):

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

   Заметим, что данные равенства объясняют смысл названий комбинаторов:

   S verSchmelzung, «сплавление»
   K Konstanz
   I Identität

4. 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:

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

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

5. 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.

6. Сформулируем определение бета-редукции на языке пред-лямбда-термов. $A \to_\beta B$, 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 $P \to_\beta P'$ и $Q = Q'$, либо $P = P'$ и $Q \to_\beta Q'$;
   - $A \equiv (\lambda x.P)$, $B \equiv (\lambda x.P')$, и $P \to_\beta P'$.

   (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.

7. Две задачи на вычисление СЗНФ при помощи нормального порядка редукции.

   (a) Постройте функцию прибавления 1 к значениям из списка в лямбда-исчислении: `let plus1 l = map (fun x -> x+1) l`. Постройте бесконечный список из нечётных чисел `[1,3,..]`. Примените функцию `plus1` к списку и получите результат в СЗНФ.

   (b) Напишите функцию вычисления суммы первых двух элементов списка: `let compute (l1::l2::ls) = (l1+l2, ls)`, примените её к результату предыдущего пункта, получите результат в СЗНФ.

8. Как мы уже разбирали, $\not\vdash x\ x : \tau$ в силу дополнительных ограничений правила

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

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

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

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

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

   (a) Покажите, что в исчислении по Карри не выполняется свойство расширения типизации (subject expansion) даже в такой формулировке: если $\vdash_к M : \sigma$, $M \twoheadrightarrow_\beta N$ и $\vdash_к N : \tau$, то необязательно, что $\sigma = \tau$.

   (b) Покажите, что в исчислении по Чёрчу свойство расширения типизации в такой формулировке также не выполняется:

   найдутся такие M, N, σ, что $\vdash_ч N : \sigma$, $M \twoheadrightarrow_\beta N$, но $\not\vdash_ч M : \sigma$.

   Но при этом в исчислении по Чёрчу выполнено свойство расширения типизации в такой формулировке:

   если $\vdash_к M : \sigma$, $M \twoheadrightarrow_\beta N$ и $\vdash_к N : \tau$, то тогда $\sigma = \tau$.

Домашнее задание №2: теорема о сильной нормализуемости просто типизированного лямбда-исчисления

1. Найдите $[\![\alpha \to \alpha]\!]$ и $[\![(\alpha \to \alpha) \to \alpha]\!]$.

2. Покажите, что $[\![\alpha]\!]$ насыщенное, и что если $\mathcal{A}$ и $\mathcal{B}$ насыщены, то $\mathcal{A} \to \mathcal{B}$ насыщенное.

3. Покажите, что SN — насыщенное (постройте полноценные рассуждения по индукции для определения).

Домашнее задание №3: 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
$[] \to [1]$	$[]$	*push* 1
$[1] \to [3; 1]$	$[]$	*push* 3
$[3; 1] \to []$	$[] \to [1; 3] \to [3]$	*pull*
$[] \to [5]$	$[3]$	*push* 5

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 σ найдётся эквивалентное σ' с поверхностными кванторами.

4. Покажите, что в предыдущем задании также имеется изоморфизм типов: существует биективная функция $\sigma \to \sigma'$, 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 HM 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
```

(a) 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 τ_α. 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?

(b) Предложите функции для печати списка и для удаления элемента списка (головы).

(c) Предложите функцию для эффективного соединения двух списков (источник для вдохновения — сложение двух чисел в столбик).

(d) Предложите функцию для эффективного выделения n-го элемента из списка.

Домашнее задание №4: Обобщённая система типов, гомотопическая теория типов, 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** : $\star \to \star$, 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) $\star \to \star \to \star$; $\star \to \textbf{unsigned}$

   (b) $\textbf{int} \to (\star \to \star)$

   (c) $(\star \to \textbf{int}) \to \star$

   (d) $\Pi x^\star.\lambda n^{\textbf{int}}.F(n, x)$, где

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

3. 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.

4. 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) коммутативность умножения;

   (b) дистрибутивность: $(a + b) \cdot c = a \cdot c + b \cdot c$;

   (c) куб суммы: $(a + 1)^3 = a^3 + 3 \cdot a^2 + 3 \cdot a + 1$.

5. Определим, что x делится на p, если обитаем тип `\Sigma (q : Nat) (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;

6. 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 непросто;

(c) 2 — единственное чётное простое число.

7. 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 : `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 $\exists k^{\mathbb{N}_0}.a + k = b$.

8. 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` $(1 \cdot 2 \cdot 3 \cdot \dots \cdot n)$ 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 : IndFin m` и `y : IndFin n`, то `x + y : IndFin (m + n)`.

Домашнее задание №5: Ещё доказательства, 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 $a_0 < a_1 < \cdots < a_n$, то список $[a_0, a_1, \ldots, a_n]$ уникальный.

   (c) Покажите, что если $n \geq 2$ и $a_k \neq a_{k+1}$ при $0 \leq k < n$, то необязательно, что список $[a_0, a_1, \ldots, a_n]$ — 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 универума 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, определявшуюся как $\lambda m.\lambda n.n\ m$.

   (d) Деление.

   (e) Вычисление факториала.

3. 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) => {?}
```

4. 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 => {?}
```

(a) Замените `{?}` в тексте выше на корректное доказательство.

(b) 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).

(c) 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`).

Домашнее задание №6: Алнебраическая топология

1. Докажите, что $\mathbb{R}^2$ и $\mathbb{R} \times [0, +\infty)$ гомотопически эквивалентны, но не гомеоморфны.
2. Докажите, что буквы `O` и `Q` гомотопически эквивалентны, но не гомеоморфны.
3. Покажите, что для любых двух мощностей α и β 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 $\{0\} \to X$ 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.

Домашнее задание №7: Аксиома выбора, теорема Диаконеску

1. Сформулируйте на языке Аренд аксиому выбора для `\Set`-ов. Покажите, что она является теоремой. Покажите, что равенство для `\Set`-ов разрешимо.
2. Определите сетоиды в Аренде. Покажите, что целые числа образуют сетоид.
3. Докажите теорему Диаконеску на Аренде с помощью сетоидов.
4. С помощью аксиомы выбора на Аренде покажите, что любая сюрьективная функция из факторизованного множества (файл стандартной библиотеки `Relation/Equivalence.ard`, тип данных `Quotient`) в факторизованное имеет частичную обратную. И наоборот, покажите, что если любая сюрьективная функция из факторизованного множества (файл стандартной библиотеки `Relation/Equivalence.ard`, тип данных `Quotient`) в факторизованное имеет частичную обратную, то выполнена аксиома выбора.