1 Домашнее задание №8: «обобщённые типовые системы»

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

   (a) Функция возведения целого числа в квадрат: `sq x = x \star x`

   Решение.

$$\frac{\vdash \mathtt{int} : \star \qquad x : \mathtt{int} \vdash x \star x : \mathtt{int} \qquad x : \mathtt{int} \vdash \mathtt{int} : \star}{\vdash (\lambda x^{\mathtt{int}}.x \star x) : (\Pi x^{\mathtt{int}}.\mathtt{int})}$$

$$\frac{\vdash \mathtt{int} : \star \qquad x : \mathtt{int} \vdash \mathtt{int} : \star}{\vdash (\Pi x^{\mathtt{int}}.\mathtt{int}) : \star}$$

□

   (b) `sizeof`

   (c) `std::map`

   `Compare`, `Allocate` опущены, там то же самое.

   Решение.

$$\dfrac{\vdash \star : \square \qquad \dfrac{\vdash \star : \square \qquad \dfrac{\vdash \star : \square \qquad \vdash \star : \square}{k : \star, v : \star \vdash \star : \square}\text{ослабл.}}{k : \star \vdash \Pi v^{\star}.\star : \square}\Pi}{\vdash \Pi k^{\star}.(\Pi v^{\star}.\star) : \square}\Pi$$

□

   (d) Монада `ST` из Хаскеля.

   Решение. Кажется, то же самое, что и `std::map`. Вся магия `ST` в `runST` — она второго kinda, но нас это не волнует. □

   (e) Пусть задано выражение рода **nonzero** : $\star \to \star$, выбрасывающее нулевой элемент из типа. Например, **nonzero unsigned** — тип положительных целых чисел. Определите, каков в коде

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

   будет тип (род) поля value.

   Решение. $\star \to \text{nonzero}\ \star$

$$\frac{\vdash \star : \square \qquad x : \star \vdash \text{nonzero}\ x : \star}{\vdash (\Pi x^{\star}.\text{nonzero}\ x) : \star}$$

$\square$

2. Приведём следующее странное рассуждение: если мы рассмотрим правый нижний дальний угол лямбда-куба, соответствующий $S = \{\langle\star, \star\rangle, \langle\star, \square\rangle, \langle\square, \star\rangle\}$, то можем заметить, что теоретически возможно существование функций, отображающих тип в значение — а потом значение в тип (например, по типу вернуть его название в строке, изменить его, а потом по изменённому названию построить другой тип).

   Поясните, почему тем не менее необходимо существование случая $\langle\square, \square\rangle$ в аксиоматике, почему всё равно мы не сможем формально построить функции рода $\Pi x^\star.F\ x$ в такой теории.

   Решение. По какому правилу мы получим $(\Pi x^\star.F\ x) : \square$?

$$\frac{\Gamma \vdash \star : \star \qquad \ldots}{\Gamma \vdash (\Pi x^\star.F\ x) : \square}\ \Pi\text{-правило}$$

   Такого не может быть, потому что если $\Gamma \vdash \star : \star$, то $\star \to \star : \star$, но $\star \to \star : \square$.

   - Очевидно по λ-правилу, аксиоме и начальному правилу не может быть.
   - Применение откладывает доказательство искомого "на потом":

$$\frac{\Gamma \vdash \varphi : (\Pi y^A.B) : s \qquad \Gamma \vdash a : A}{\Gamma \vdash (\varphi\ a) \equiv \Pi q x^\star.F\ x : (B[y := A]) \equiv \square}\ \text{применение}$$
$$B \equiv \Pi x^\star.F'\ y\ x$$

     Нужно найти тип B, что сводится к исходной задаче.
   - β-редукция из преобразования не поможет — не работает на S.
   - Для ослабления нужно опять доказать искомое.

$\square$

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

   (a) **int** $\to (\star \to \star)$

```
#define bruh(n, t) std::array<t, n>
```

   (b) $(\star \to$ **int**$) \to \star$

```
template<template <typename> typename T>
    requires std::is_same_v<
        std::decay_t<decltype(T<std::nullptr_t>::value)>, // no way to check template
```

```
        int>
class answer
{};

template<typename T>
struct sizeof_v
{
    static constexpr int value = sizeof(T);
};
```

(c) $\Pi x^{\star}.n^{\mathbf{int}}.F(n,x)$, где

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

Решение.

```
template<typename x, unsigned n>
struct answer
{
    static constexpr auto get(x const&)
    {
        return answer<x, n-1>::get;
    }
};

template<typename x>
struct answer<x, 0>
{
    static constexpr int get()
    {
        return 0;
    }
};

#include <iostream>

int main()
{
    std::cout << answer<int, 2>::get(0)(1)() << std::endl; // prints 0
}
```

□

4. Аналогично типу Π, мы можем ввести тип Σ, соответствующий квантору суще-

ствования в смысле изоморфизма Карри-Ховарда.

(a) Определите правила вывода для Σ в обобщённой типовой системе (воспользуйтесь правилами для экзистенциальных типов в системе F).

(b) Укажите способ выразить Σ через Π (также воспользуйтесь идеями для системы F).

$$\text{pack}\,\tau, M \text{ to } \Sigma\alpha.\sigma = \lambda\beta^*.\lambda x^{\Pi\alpha^*.\sigma\to\beta}.x\ \tau\ M : \Pi\beta^*.\,(\Pi\alpha^*.\sigma \to \beta) \to \beta$$

5. Рассмотрим классы типов в Хаскеле (например, `Num`). Каким образом их можно представить в обобщённой типовой системе? Как формализовать запись типа функции `f :: Num a => a -> a`?