

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: $\text{sq } x = x \star x$

Решение.

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

$$\frac{\vdash \text{int} : \star \quad x : \text{int} \vdash \text{int} : \star}{\vdash (\Pi x^{\text{int}}. \text{int}) : \star}$$

□

(b) `sizeof`

(c) `std::map`

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

Решение.

$$\frac{\vdash \star : \square \quad \frac{\vdash \star : \square \quad \vdash \star : \square}{k : \star, v : \star \vdash \star : \square} \text{ослабл.} \quad \Pi}{\vdash \star : \square \quad k : \star \vdash \Pi v^*. \star : \square} \Pi \quad \Pi$$

$$\vdash \Pi k^*. (\Pi v^*. \star) : \square$$

□

(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`** : $\star \rightarrow \star$, 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. $\star \rightarrow \text{nonzero } \star$

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

□

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 $(\Pi x^\star. F x) : \square$?

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

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

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

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

$$B \equiv \Pi x^\star. F' y x$$

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.

□

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

(a) `int` \rightarrow `($\star \rightarrow \star$)`

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

(b) `($\star \rightarrow$ int) \rightarrow \star`

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

```

4         int>
5     class answer
6     {};
7
8     template<typename T>
9     struct sizeof_v
10    {
11        static constexpr int value = sizeof(T);
12    };

```

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

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

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).

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

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`?