Функциональное программирование
Лекция 13. Алгоритм вывода типов

Денис Николаевич Москвин

СПбГУ, факультет МКН,
бакалавриат «Современное программирование», 2 курс

04.12.2025

План лекции

1. Главный тип
2. Подстановка типа и унификация
3. Теорема Хиндли-Милнера
4. `let`-полиморфизм и типы высших рангов
5. К практике

План лекции

1. Главный тип
2. Подстановка типа и унификация
3. Теорема Хиндли-Милнера
4. `let`-полиморфизм и типы высших рангов
5. К практике

Система $\lambda_{\to}$ а ля Карри

Предтермы	Редукция
$\Lambda ::= V \mid M\,N \mid \lambda x.\,M$	$(\lambda x.\,M)\,N \rightsquigarrow_{\beta} M[x := N]$
Типы	**Типизация**
$\mathbb{T} ::= \mathbb{V} \mid A \to B$	$\dfrac{x^A \in \Gamma}{\Gamma \vdash x : A}$
	$\dfrac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash M\,N : B}$
Контексты	
$\Gamma ::= \varnothing \mid \Gamma, x^A$	$\dfrac{\Gamma, x^A \vdash M : B}{\Gamma \vdash \lambda x.\,M : A \to B}$

Здесь $V = \{a, b, \ldots\}$, $\mathbb{V} = \{\alpha, \beta, \ldots\}$ и
$x \in V;\; M, N \in \Lambda;\; A, B \in \mathbb{T}$.

Система $\lambda_{\to}$ а ля Чёрч

Предтермы	Редукция
$\Lambda_{\mathbb{T}} ::= V \mid M\,N \mid \lambda x^{A}.\,M$	$(\lambda x^{A}.\,M)\,N \rightsquigarrow_{\beta} M[x := N]$
Типы	**Типизация**
$\mathbb{T} ::= \mathbb{V} \mid A \to B$	$\dfrac{x^{A} \in \Gamma}{\Gamma \vdash x : A}$
Контексты	$\dfrac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash M\,N : B}$
$\Gamma ::= \varnothing \mid \Gamma, x^{A}$	$\dfrac{\Gamma, x^{A} \vdash M : B}{\Gamma \vdash \lambda x^{A}.\,M : A \to B}$

Здесь $V = \{a, b, \ldots\}$, $\mathbb{V} = \{\alpha, \beta, \ldots\}$ и
$x \in V$; $M, N \in \Lambda_{\mathbb{T}}$; $A, B \in \mathbb{T}$.

Главный тип (principal type)

- Для систем Карри и Черча верна *лемма подстановки типа*: $\Gamma \vdash M : B \;\Rightarrow\; [\alpha := A]\Gamma \vdash [\alpha := A]\, M : [\alpha := A]B$.
- В версии Чёрча $\lambda_{\to}$ термы атрибутированы типами, поэтому тип терма единственен:

$$\lambda f^{\sigma\to\tau\to\rho}\, g^{\sigma\to\tau}\, z^{\sigma}.\, f\, z\, (g\, z) : (\sigma\to\tau\to\rho)\to(\sigma\to\tau)\to\sigma\to\rho$$

$$\lambda f^{\sigma\to\tau\to\sigma}\, g^{\sigma\to\tau}\, z^{\sigma}.\, f\, z\, (g\, z) : (\sigma\to\tau\to\sigma)\to(\sigma\to\tau)\to\sigma\to\sigma$$

$$\lambda f^{(\tau\to\rho)\to\tau\to\rho}\, g^{(\tau\to\rho)\to\tau}\, z^{\tau\to\rho} :$$
$$((\tau\to\rho)\to\tau\to\rho)\to((\tau\to\rho)\to\tau)\to(\tau\to\rho)\to\rho$$

- Любой из этих типов можно приписать терму $\mathbf{S} \equiv \lambda f\, g\, z.\, f\, z\, (g\, z)$ в версии Карри.
- Однако, первый «лучше» в том смысле, что остальные получаются из него подстановками типа вместо типовых переменных. Он называется ***главным (principal)***.

Вывод главного типа (пример для комбинатора)

$$\lambda x\, y.\, y\, (\lambda z.\, y\, x) \qquad \lambda x^{\alpha}\, y^{\beta}.\, \underbrace{y^{\beta}\, (\lambda z^{\gamma}.\, \overbrace{y^{\beta}\, x^{\alpha}}^{\delta})}_{\varepsilon}$$

1. Припишем типовую (мета-)переменную всем термовым переменным: $x^{\alpha}, y^{\beta}, z^{\gamma}$.
2. Припишем типовую (мета-)переменную всем *аппликативным* подтермам: $y\, x : \delta,\ y\, (\lambda z.\, y\, x) : \varepsilon$.
3. Выпишем уравнения (ограничения) на типы, необходимые для типизируемости терма: $\beta \sim \alpha \to \delta, \quad \beta \sim (\gamma \to \delta) \to \varepsilon$.
4. Найдём *главный унификатор* для типовых переменных (подстановку), дающий решения уравнений: $\alpha := \gamma \to \delta, \quad \beta := (\gamma \to \delta) \to \varepsilon, \quad \delta := \varepsilon$.
5. Главный тип $\lambda x\, y.\, y\, (\lambda z.\, y\, x) : (\gamma \to \varepsilon) \to ((\gamma \to \varepsilon) \to \varepsilon) \to \varepsilon$.

План лекции

1. Главный тип
2. Подстановка типа и унификация
3. Теорема Хиндли-Милнера
4. `let`-полиморфизм и типы высших рангов
5. К практике

Подстановка типа

Определение

Подстановка типа — это операция $S : \mathbb{T} \to \mathbb{T}$, такая что

$$S(A \to B) \equiv S(A) \to S(B)$$

- Обычно подстановка тождественна на всех переменных, кроме конечного носителя $\sup(S) = \{\alpha \mid S(\alpha) \not\equiv \alpha\}$.
- Пример подстановки $S = [\alpha := \gamma \to \beta, \beta := \alpha \to \gamma]$.
- Подстановка выполняется *параллельно*

$$\begin{aligned} S(\alpha \to \beta \to \gamma) &= [\alpha := \gamma \to \beta, \beta := \alpha \to \gamma](\alpha \to \beta \to \gamma) \\ &= (\gamma \to \beta) \to (\alpha \to \gamma) \to \gamma \end{aligned}$$

Подстановка типа

Определение

Подстановка типа — это операция $S : \mathbb{T} \to \mathbb{T}$, такая что

$$S(A \to B) \;\equiv\; S(A) \to S(B)$$

- Обычно подстановка тождественна на всех переменных, кроме конечного носителя $\sup(S) = \{\alpha \mid S(\alpha) \not\equiv \alpha\}$.
- Пример подстановки $S = [\alpha := \gamma \to \beta, \beta := \alpha \to \gamma]$.
- Подстановка выполняется *параллельно*

$$\begin{aligned} S(\alpha \to \beta \to \gamma) &= [\alpha := \gamma \to \beta, \beta := \alpha \to \gamma](\alpha \to \beta \to \gamma) \\ &= (\gamma \to \beta) \to (\alpha \to \gamma) \to \gamma \end{aligned}$$

Тождественна ли $[\beta := \alpha]([\alpha := \beta]\, C)$? Эквивалентна ли она $[\alpha := \beta]([\beta := \alpha]\, C)$? $[\beta := \alpha, \alpha := \beta]\, C)$?

Лемма подстановки и ее следствия

Для подстановок с одноэлементным носителем верна

Лемма подстановки

Для $\alpha_1 \neq \alpha_2$ и $\alpha_1 \notin FV(A_2)$ верно равенство

$$[\alpha_2 := A_2]\,([\alpha_1 := A_1]\,B) \;\equiv\; [\alpha_1 := [\alpha_2 := A_2]\,A_1]\,([\alpha_2 := A_2]\,B)$$

- Если $\alpha_1 \in FV(A_2)$, то слева эти вхождения α_1 не будут ни на что замещаться, а справа — могут.
- Доказательство индукцией по структуре B. Нетривиальные случаи: листья α_1 и α_2.
- Если $\alpha_1 \notin FV(A_2)$ и $\alpha_2 \notin FV(A_1)$, то элементарные подстановки коммутируют. Лемма дает возможность по первому условию достичь второго. Для любого C

$$\begin{aligned} [\alpha_2 := A_2]\,([\alpha_1 := A_1]\,C) &\equiv [\alpha_1 := A_1']\,([\alpha_2 := A_2]\,C) \\ &\equiv [\alpha_1 := A_1', \alpha_2 := A_2]\,C \end{aligned}$$

Композиция подстановок

Определение

Композиция двух подстановок — подстановка с носителем, являющимся объединением их носителей*, над которым последовательно выполнены обе подстановки.

Для $S = [\alpha := \gamma \to \beta, \beta := \alpha \to \gamma]$ и $T = [\alpha := \beta \to \gamma, \gamma := \beta]$

$$\begin{aligned} T \circ S &= [\alpha := T(S(\alpha)), \beta := T(S(\beta)), \gamma := T(S(\gamma))] \\ &= [\alpha := \beta \to \beta, \beta := (\beta \to \gamma) \to \beta, \gamma := \beta] \end{aligned}$$

Утверждение

Подстановки образуют моноид относительно $\circ$ с $[\,]$ в роли нейтрального элемента.

* Найдите $[\beta := \alpha] \circ [\alpha := \beta]$.

Лемма о композиции подстановок

Если подстановка определена как композиция элементарных

$$[\alpha_1 := A'_1, \ldots, \alpha_n := A'_n] \equiv [\alpha_n := A_n] \circ \ldots \circ [\alpha_1 := A_1]$$

причем $\forall i \leqslant j\ \alpha_i \notin FV(A_j)$, то

$$\forall i, j\ \alpha_i \notin FV(A'_j)$$

Доказательство. Индукция с использованием Леммы подстановки. ■

Унификатор

Определение

Унификатор для типов A и B — это подстановка S, такая что $S(A) \equiv S(B)$.

Пример

Пусть $A = \alpha \to \beta \to \gamma$ и $B = (\delta \to \varepsilon) \to \zeta$.
Их унификатор

$$S = [\alpha := \delta \to \varepsilon, \zeta := \beta \to \gamma]$$

Действительно, в результате этой подстановки получаем и из B и из A один и тот же тип

$$S(A) \equiv S(B) \equiv (\delta \to \varepsilon) \to \beta \to \gamma$$

Главный унификатор

Определение

S — это ***главный унификатор*** для A и B, если для любого другого унификатора S' существует подстановка T, такая что

$$S' \equiv T \circ S$$

Пример

Для $A = \alpha \to \beta \to \gamma$ и $B = (\delta \to \varepsilon) \to \zeta$ главный унификатор

$$\begin{aligned}
S &= [\alpha := \delta \to \varepsilon, \zeta := \beta \to \gamma] \\
\\
S' &= [\alpha := \delta \to \color{red}{\beta}, \zeta := \beta \to \gamma] \\
S' &= [\varepsilon := \beta] \circ S \\
\\
S'' &= [\alpha := \color{red}{(\gamma \to \beta)} \to \varepsilon, \zeta := \beta \to \gamma] \\
S'' &= [\delta := \gamma \to \beta] \circ S
\end{aligned}$$

Поиск унификатора

Попробуем унифицировать типы

$$\begin{array}{lcl} A & = & \alpha \to \beta \to \alpha \\ B & = & \gamma \to \delta \end{array}$$

Для построения унификатора нужно соединить подстановки $[\alpha := \gamma]$ и $[\delta := \beta \to \alpha]$. Рассмотрим композиции

$$\begin{array}{lclcl} S_1 & = & [\alpha := \gamma] \circ [\delta := \beta \to \alpha] & = & [\delta := \beta \to \gamma, \alpha := \gamma] \\ S_2 & = & [\delta := \beta \to \alpha] \circ [\alpha := \gamma] & = & [\delta := \beta \to \alpha, \alpha := \gamma] \end{array}$$

Одна из подстановок — унификатор, другая — нет:

$$\begin{array}{lcllcl} S_1(A) & \equiv & \gamma \to \beta \to \gamma & \quad S_1(B) & \equiv & \gamma \to \beta \to \gamma \\ S_2(A) & \equiv & \gamma \to \beta \to \gamma & \quad S_2(B) & \equiv & \gamma \to \beta \to \alpha \end{array}$$

Мораль: выделив одну из элементарных подстановок, следует тут же сделать ее повсюду.

Теорема унификации

Теорема унификации (Робинсон, 1965; Martelli, Montanari, 1982)

Существует алгоритм унификации $\mathcal{U}$, который для заданных типов A и B возвращает:

- главный унификатор S для A и B, если A и B могут быть унифицированы;
- сообщение об ошибке в противном случае.

- Алгоритм $\mathcal{U}(A, B)$ позволяет искать «минимальное» решение уравнения на типы $A \sim B$.
- Ключевой момент всех рассуждений про унификацию:

$$A_1 \to A_2 \equiv B_1 \to B_2 \;\Leftrightarrow\; A_1 \equiv B_1 \;\wedge\; A_2 \equiv B_2$$

Алгоритм унификации $\mathcal{U}$

$$\begin{array}{lcl}
\mathcal{U}(\alpha,\alpha) & = & [\,] \\
\mathcal{U}(\alpha,B) \mid \alpha \in FV(B) & = & \text{ошибка} \\
\mathcal{U}(\alpha,B) \mid \alpha \notin FV(B) & = & [\alpha := B] \\
\mathcal{U}(A_1 \to A_2, \alpha) & = & \mathcal{U}(\alpha, A_1 \to A_2) \\
\mathcal{U}(A_1 \to A_2, B_1 \to B_2) & = & \mathcal{U}\,(\mathcal{U}_2 A_1, \mathcal{U}_2 B_1) \circ \mathcal{U}_2 \\
 & & \text{where } \mathcal{U}_2 = \mathcal{U}(A_2, B_2)
\end{array}$$

- $\mathcal{U}(A,B)$ завершается. Деревья типа конечны и количество типовых переменных сокращается через конечное число шагов.
- $\mathcal{U}(A,B)$ унифицирует. По индукции; используем, что если S унифицирует (A,B), то $S \circ [\alpha := C]$ унифицирует $(A \to \alpha, B \to C)$.
- $\mathcal{U}(A,B)$ даёт главный унификатор. По индукции; см. TAPL (глава 22.4) [Pie02] или LCwT (глава 4.4) [Bar92].

Алгоритм унификации $\mathcal{U}$: пример

$$
\begin{array}{lcl}
\mathcal{U}(\alpha, \alpha) & = & [\,] \\
\mathcal{U}(\alpha, B) \mid \alpha \in FV(B) & = & \text{ошибка} \\
\mathcal{U}(\alpha, B) \mid \alpha \notin FV(B) & = & [\alpha := B] \\
\mathcal{U}(A_1 \to A_2, \alpha) & = & \mathcal{U}(\alpha, A_1 \to A_2) \\
\mathcal{U}(A_1 \to A_2, B_1 \to B_2) & = & \mathcal{U}\,(\mathcal{U}(A_2, B_2)A_1, \mathcal{U}(A_2, B_2)B_1) \circ \\
 & & \mathcal{U}(A_2, B_2)
\end{array}
$$

Для $\lambda x\, y.\, y\, (\lambda z.\, y\, x)$ система уравнений на типы имела вид $E = \{\beta \sim (\gamma \to \delta) \to \varepsilon, \beta \sim \alpha \to \delta\}$. Алгоритм $\mathcal{U}$ даёт:

$$
\begin{array}{lcl}
\mathcal{U}(E) & = & \mathcal{U}(\beta \to \beta, ((\gamma \to \delta) \to \varepsilon) \to (\alpha \to \delta)) \\
 & = & \mathcal{U}(\mathcal{U}(\beta, \alpha \to \delta)\beta, \mathcal{U}(\beta, \alpha \to \delta)(\gamma \to \delta) \to \varepsilon) \circ \mathcal{U}(\beta, \alpha \to \delta) \\
 & = & \mathcal{U}(\alpha \to \delta, (\gamma \to \delta) \to \varepsilon) \circ [\beta := \alpha \to \delta] \\
 & = & [\alpha := \gamma \to \varepsilon] \circ [\delta := \varepsilon] \circ [\beta := \alpha \to \delta] \\
 & = & [\beta := (\gamma \to \varepsilon) \to \varepsilon, \delta := \varepsilon, \alpha := \gamma \to \varepsilon]
\end{array}
$$

Домашнее задание

- Проследите за изменениями в работе алгоритма $\mathcal{U}$, при перестановке элементов в E:
  $E = \{\beta \sim \alpha \to \delta, \beta \sim (\gamma \to \delta) \to \varepsilon\}$
- Изменится ли что-то в этом случае, и, если изменится, то что?

План лекции

1. Главный тип
2. Подстановка типа и унификация
3. Теорема Хиндли-Милнера
4. `let`-полиморфизм и типы высших рангов
5. К практике

Теорема о существовании системы ограничений

- Наша первая цель — построить систему ограничений на типы для терма M (возможно незамкнутого).
- Для типизации таких термов необходим контекст Γ, в котором объявляются типы всех свободных переменных.
- Для S, унифицирующей систему уравнений на типы $E = \{A_1 \sim B_1, \ldots, A_n \sim B_n\}$, введем обозначение $S \vDash E$.

Теорема о существовании системы ограничений

Для любых терма $M \in \Lambda$, контекста Γ ($\mathrm{FV}(M) \subseteq \mathrm{dom}(\Gamma)$) и типа $A \in \mathbb{T}$ существует конечное множество уравнений на типы $E = E(\Gamma, M, A)$, такое что для любой подстановки S:

- $S \vDash E(\Gamma, M, A) \;\Rightarrow\; S(\Gamma) \vdash M : S(A)$;
- $S(\Gamma) \vdash M : S(A) \;\Rightarrow\; S' \vDash E(\Gamma, M, A)$, для некоторой S', имеющего тот же эффект, что и S, на типовых переменных в A и Γ.

- Оговорка про S' нужна, поскольку в E могут быть типовые переменные, которых нет в A и Γ.

Алгоритм построения системы ограничений

Алгоритм построения системы ограничений E

$$
\begin{array}{rcl}
E(\Gamma, x, A) & = & \{A \sim \Gamma(x)\} \\
E(\Gamma, P\,Q, A) & = & E(\Gamma, P, \alpha \to A) \cup E(\Gamma, Q, \alpha) \\
E(\Gamma, \lambda x.\,P, A) & = & E((\Gamma, x^{\alpha}), P, \beta) \cup \{\alpha \to \beta \sim A\}
\end{array}
$$

- В первом равенстве контекст Γ рассматривается как функция из множества переменных в множество типов.
- Переменные α во втором и третьем равенствах и β в третьем всякий раз должны быть «свежими».
- Самостоятельно постройте системы ограничений для следующих троек (Γ, M, A)

$$
\begin{array}{rcl}
E(x^{\gamma \to \delta}, x, \quad \varepsilon) & = & ??? \\
E(x^{\gamma \to \delta}, x\,x, \quad \varepsilon) & = & ??? \\
E(x^{\gamma \to \delta}, \lambda x.\,x, \varepsilon) & = & ???
\end{array}
$$

Главная пара (Principal Pair)

Определение

Для $M \in \Lambda$ ***главной парой*** называют пару (Γ, A), такую что

- $\Gamma \vdash M : A$
- $\Gamma' \vdash M : A' \;\Rightarrow\; \exists S \; [S(\Gamma) \subseteq \Gamma' \;\wedge\; S(A) \equiv A']$

Пример: $M = \lambda y.\, y\, x$

$$PP(\lambda y.\, y\, x) \;=\; (x^{\alpha}, (\alpha \to \beta) \to \beta)$$

$$\begin{array}{rcl}
x^{\alpha} & \vdash & \lambda y.\, y\, x : (\alpha \to \beta) \to \beta \\
x^{\beta} & \vdash & \lambda y.\, y\, x : (\beta \to \beta) \to \beta \\
x^{\beta \to \gamma} & \vdash & \lambda y.\, y\, x : ((\beta \to \gamma) \to \beta) \to \beta \\
x^{\alpha}, z^{\beta} & \vdash & \lambda y.\, y\, x : (\alpha \to \beta) \to \beta
\end{array}$$

Теорема Хиндли – Милнера

Существует алгоритм PP, возвращающий для $M \in \Lambda$

- главную пару (Γ, A), если M имеет тип;
- сообщение об ошибке в противном случае.

Пусть $FV(M) = \{x_1, \ldots, x_n\}$. Выберем произвольные различные переменные $\alpha_0, \ldots, \alpha_n$ и сконструируем $\Gamma_0 = \{x_1^{\alpha_1}, \ldots, x_n^{\alpha_n}\}$.

Алгоритм PP

$$\begin{array}{lcl} PP(M) \mid U(E(\Gamma_0, M, \alpha_0)) \equiv \text{ошибка} & = & \text{ошибка} \\ PP(M) \mid U(E(\Gamma_0, M, \alpha_0)) \equiv S & = & (S(\Gamma_0), S(\alpha_0)) \end{array}$$

Стартуем с произвольных переменных типа, приписанных свободным переменным типизируемого терма M и всему терму.

Главный тип (Principal Type)

Для комбинаторов (термов без свободных переменных) необходимый контекст пуст, и мы приходим к понятию главного типа.

Определение

Для $M \in \Lambda^0$ ***главным типом*** называют тип A, такой что

- $\vdash M : A$
- $\vdash M : A' \;\Rightarrow\; \exists S \;\left[S(A) \equiv A'\right]$

Следствие теоремы Хиндли – Милнера

Существует алгоритм PT, возвращающий для $M \in \Lambda^0$

- главный тип A, если M имеет тип;
- сообщение об ошибке в противном случае.

План лекции

1. Главный тип
2. Подстановка типа и унификация
3. Теорема Хиндли-Милнера
4. `let`-полиморфизм и типы высших рангов
5. К практике

Расширение на произвольные типовые конструкторы

- Алгоритм Хиндли-Милнера легко расширить с простых типов, где
  - листья образованы только переменными,
  - узлы образованы только `(->) :: * -> * -> *`,

  на произвольные конструкторы типа:
  - листья — конструкторы типа кайнда `*`;
  - узлы — конструкторы типа любых стрелочных кайндов;
  - узлы — полиморфные, например, `m a` или `t m a`.
- Решите следующие уравнения на типы

```
Char -> a  ~  b -> Bool -> b

c Int a -> Char  ~  (b,Char) -> a

Alternative f => f (Bool -> a) ~ Monad m => m (b -> b)

t (State s Int) Char ~ MaybeT m a
```

Где Хиндли-Милнер не справляется?

Есть одно тонкое место, в котором алгоритм Хиндли-Милнера не работает:

```
GHCi> :t \f -> (f 'z', f True)
Couldn't match expected type `Char' with actual type `Bool'
  In the first argument of `f', namely `True'
  In the expression: f True
```

Почему это плохо?

Где Хиндли-Милнер не справляется?

Есть одно тонкое место, в котором алгоритм Хиндли-Милнера не работает:

```
GHCi> :t \f -> (f 'z', f True)
Couldn't match expected type `Char' with actual type `Bool'
  In the first argument of `f', namely `True'
  In the expression: f True
```

Почему это плохо? Потому что вместо `f` мы можем передать полиморфную функцию

```
GHCi>(\f -> (f 'z', f True)) (\x -> x)
Couldn't match expected type `Char' with actual type `Bool'
  In the first argument of `f', namely `True'
  In the expression: f True
```

Для решения этой проблемы нужен более точный контроль за местом применения функции.

let-полиморфизм

В Haskell и языках семейства ML `let...in...` рассматривают как примитив, а не как синтаксический сахар для лямбды.

```
GHCi> (\f -> (f 'z', f True)) (\x -> x)
Couldn't match expected type `Char' with actual type `Bool'

GHCi> let f = \x -> x in (f 'z', f True)
('z',True)
```

В последнем случае полиморфный тип
`f :: forall a. a -> a` мономорфизируется индивидуально для каждого вхождения `f`.

Типы второго ранга

Однако `let`-полиморфизм не панацея.

```
GHCi> let f g = (g 'z', g True) in f id
Couldn't match expected type `Char' with actual type `Bool'
  In the first argument of `g', namely `True'
  In the expression: g True
```

В более богатой системе это возможно, нужно лишь явно указать тип, поскольку вывод типов для систем высших рангов (> 2) неразрешим.

```
GHCi> :set -XRankNTypes
GHCi> let { f :: (forall a. a -> a) -> (Char, Bool);
f g = (g 'z', g True) } in f id
('z',True)
```

Расширение `RankNTypes` входит в `GHC2021`.

План лекции

1. Главный тип
2. Подстановка типа и унификация
3. Теорема Хиндли-Милнера
4. `let`-полиморфизм и типы высших рангов
5. К практике

Задание для практики (и на дом)

1. Реализуйте алгоритм $\mathcal{U}$ на Haskell.
2. Реализуйте алгоритм $\mathcal{E}$ на Haskell.
3. Реализуйте алгоритм $\mathcal{PP}$ на Haskell.

Определение лямбда-терма

- Лямбда-термы можно закодировать так

```
type Symb = String

infixl 4 :@

data Expr
        = Var Symb
        | Expr :@ Expr
        | Lam Symb Expr
        deriving (Eq, Show)
```

- Например, выражение
  `(Lam "x" $ Lam "y" $ Var "x") :@ (Lam "z" $ Var "z")`
  кодирует терм $(\lambda x\, y.\, x)\, (\lambda z.\, z)$.

Свободные переменные

Свободные переменные терма

```
freeVars :: Expr -> [Symb]
```

Попробуйте написать реализацию.

Свободные переменные

Свободные переменные терма

```
freeVars :: Expr -> [Symb]
```

Попробуйте написать реализацию.

Реализация

```
freeVars :: Expr -> [Symb]
freeVars (Var v)      = [v]
freeVars (t1 :@ t2) = undefined
freeVars (Lam v t)  = undefined
```

Поскольку мы используем список для кодирования множества, следует обеспечить уникальность.

Определение типа

- Типы можно закодировать так

```haskell
infixr 3 :->

data Type
    = TVar Symb
    | Type :-> Type
    deriving (Eq, Show)
```

- Например, выражение `(TVar "a" :-> TVar "b") :-> TVar "c"` кодирует тип $(a \to b) \to c$, а выражение `TVar "a" :-> TVar "b" :-> TVar "c"` кодирует тип $a \to b \to c$.

Контексты

- Контексты можно закодировать так

```
newtype Env = Env [(Symb, Type)]
  deriving (Eq,Show)
```

- Полезно реализовать расширение контекста в виде сервисной функции

```
extendEnv :: Env -> Symb -> Type -> Env
extendEnv (Env env) s t = Env $ (s,t) : env
```

Здесь уникальность имен вредна — нужно уметь разбирать термы с повторяющимися именами связанных переменных, например, $\lambda x.\, \lambda x.\, x$.

Литература

Б. Пирс.
Типы в языках программирования.
Лямбда пресс, Добросвет, Москва, 2012.

H.P. Barendregt.
Lambda calculi with types.
In *Handbook of Logic in Computer Science*, pages 117–309.
Oxford University Press, 1992.

Benjamin C. Pierce.
Types and Programming Languages.
MIT Press, 2002.