Функциональное программирование
Лекция 13. Алгоритм вывода типов

Денис Николаевич Москвин

СПбАУ РАН

05.12.2017

План лекции

План лекции

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

<table>
<tr><td>

Предтермы

$$\begin{array}{rcl}\Lambda ::= & & V \\ & \mid & M\,N \\ & \mid & \lambda x.\,M\end{array}$$

</td><td>

Редукция

$$(\lambda x.\,M)\,N \to_\beta M[x := N]$$

</td></tr>
<tr><td>

Типы

$$\begin{array}{rcl}\mathbb{T} ::= & & \mathbb{V} \\ & \mid & \sigma \to \tau\end{array}$$

Контексты

$$\begin{array}{rcl}\Gamma ::= & & \varnothing \\ & \mid & \Gamma, x:\sigma\end{array}$$

</td><td>

Типизация

$$\frac{x:\sigma \in \Gamma}{\Gamma \vdash x:\sigma}$$

$$\frac{\Gamma \vdash M:\sigma\to\tau \quad \Gamma \vdash N:\sigma}{\Gamma \vdash (M\,N):\tau}$$

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

</td></tr>
</table>

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

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

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

Главный тип (principle type)

- Для систем Карри и Черча верна *лемма подстановки типа*: $\Gamma \vdash M : \sigma \;\Rightarrow\; [\alpha := \tau]\Gamma \vdash [\alpha := \tau]\, M : [\alpha := \tau]\sigma$.
- 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 $\boldsymbol{S} \equiv \lambda f\, g\, z.\, f\, z\, (g\, z)$ в версии Карри.
- Однако, первый «лучше» в том смысле, что остальные получаются из него подстановкой типа вместо типовой переменной. Он называется ***главным (principle)***.

Вывод главного типа (пример)

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

1. Присвоим типовую (мета-)переменную всем термовым переменным: $x^{\alpha}, y^{\beta}, z^{\gamma}$.
2. Присвоим 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

Подстановка типа

Определение

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

$$S(\sigma \to \tau) \equiv S(\sigma) \to S(\tau)$$

- Обычно подстановка тождественна на всех типовых переменных, кроме конечного носителя $\mathrm{sup}(S) = \{\alpha \mid S(\alpha) \not\equiv \alpha\}$.
- Пример подстановки $S = [\alpha := \gamma \to \beta, \beta := \alpha \to \gamma]$.
- Тождественную подстановку (с пустым носителем) обозначают $[\,]$.
- Подстановка выполняется *параллельно*; для $\tau = \alpha \to \beta \to \gamma$

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

Композиция подстановок

Определение

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

Для

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

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

Подстановки образуют моноид относительно $\circ$ с единицей $[]$.
(проверьте этот факт самостоятельно)

Унификатор

Определение

Унификатор для типов σ и τ — это подстановка S, такая что $S(\sigma) \;\equiv\; S(\tau)$.

Пример

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

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

Действительно, в результате этой подстановки получаем и из τ и из σ один и тот же тип

$$S(\sigma) \;\equiv\; S(\tau) \;\equiv\; (\delta \to \varepsilon) \to \beta \to \gamma$$

Поиск унификатора

Попробуем унифицировать типы

$$\begin{array}{rcl} \sigma & = & \alpha \to \beta \to \alpha \\ \tau & = & \gamma \to \delta \end{array}$$

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

$$\begin{array}{rclcl} S' & = & [\alpha := \gamma] \circ [\delta := \beta \to \alpha] & = & [\alpha := \gamma, \delta := \beta \to \gamma] \\ S'' & = & [\delta := \beta \to \alpha] \circ [\alpha := \gamma] & = & [\delta := \beta \to \alpha, \alpha := \gamma] \end{array}$$

Одна из подстановок — унификатор, другая — нет:

$$\begin{array}{rclcrcl} S'(\sigma) & \equiv & \gamma \to \beta \to \gamma & \qquad & S'(\tau) & \equiv & \gamma \to \beta \to \gamma \\ S''(\sigma) & \equiv & \gamma \to \beta \to \gamma & & S''(\tau) & \equiv & \gamma \to \beta \to \alpha \end{array}$$

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

Главный унификатор

Определение

Унификатор S — это ***главный унификатор*** для σ и τ, если для любого другого унификатора S' существует подстановка T, такая что

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

Пример

Для $\sigma = \alpha \to \beta \to \alpha$ и $\tau = \gamma \to \delta$ главный унификатор

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

$$S' = [\alpha := \gamma, \beta := \varepsilon \to \varepsilon, \delta := (\varepsilon \to \varepsilon) \to \gamma]$$

$$S' = [\beta := \varepsilon \to \varepsilon] \circ S$$

Теорема унификации (Робинсон, 1965)

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

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

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

$$\sigma_1 \to \sigma_2 \equiv \tau_1 \to \tau_2 \;\Leftrightarrow\; \sigma_1 \equiv \tau_1 \;\wedge\; \sigma_2 \equiv \tau_2$$

Вывод типов: алгоритм унификации $\mathfrak{U}$

Алгоритм унификации $\mathfrak{U}$

$$
\begin{array}{lcl}
\mathfrak{U}(\alpha, \alpha) & = & [\,] \\
\mathfrak{U}(\alpha, \tau) \mid \alpha \in FV(\tau) & = & \text{ошибка} \\
\mathfrak{U}(\alpha, \tau) \mid \alpha \notin FV(\tau) & = & [\alpha := \tau] \\
\mathfrak{U}(\sigma_1 \to \sigma_2, \alpha) & = & \mathfrak{U}(\alpha, \sigma_1 \to \sigma_2) \\
\mathfrak{U}(\sigma_1 \to \sigma_2, \tau_1 \to \tau_2) & = & \mathfrak{U}\,(\mathfrak{U}_2\sigma_1, \mathfrak{U}_2\tau_1) \circ \mathfrak{U}_2 \\
 & & \text{where } \mathfrak{U}_2 = \mathfrak{U}(\sigma_2, \tau_2)
\end{array}
$$

- $\mathfrak{U}(\sigma, \tau)$ завершается. Деревья типа конечны и количество типовых переменных сокращается на 1 через конечное число шагов.
- $\mathfrak{U}(\sigma, \tau)$ унифицирует. По индукции; используем, что если S унифицирует (σ, τ), то $S \circ [\alpha := \rho]$ унифицирует $(\sigma \to \alpha, \tau \to \rho)$.
- $\mathfrak{U}(\sigma, \tau)$ даёт главный унификатор. По индукции; см. TAPL (глава 22.4) [Pie02] или LCwT (глава 4.4) [Bar92].

Алгоритм унификации $\mathcal{U}$: пример

$$
\begin{array}{lcl}
\mathcal{U}(\alpha,\alpha) & = & [\,] \\
\mathcal{U}(\alpha,\tau) \mid \alpha \in FV(\tau) & = & \text{ошибка} \\
\mathcal{U}(\alpha,\tau) \mid \alpha \notin FV(\tau) & = & [\alpha := \tau] \\
\mathcal{U}(\sigma_1 \to \sigma_2, \alpha) & = & \mathcal{U}(\alpha, \sigma_1 \to \sigma_2) \\
\mathcal{U}(\sigma_1 \to \sigma_2, \tau_1 \to \tau_2) & = & \mathcal{U}\left(\mathcal{U}(\sigma_2,\tau_2)\sigma_1, \mathcal{U}(\sigma_2,\tau_2)\tau_1\right) \circ \mathcal{U}(\sigma_2,\tau_2)
\end{array}
$$

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:

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

Домашнее задание

- Проследите за изменениями в работе алгоритма 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

- 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 подстановки S, унифицирующей систему уравнений на типы $E = \{\sigma_1 \sim \tau_1, \ldots, \sigma_n \sim \tau_n\}$, введём обозначение $S \models E$.

Теорема о существовании системы ограничений

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

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

Алгоритм построения системы ограничений E

$$\begin{array}{lcl} E(\Gamma, x, \sigma) & = & \{\sigma \sim \Gamma(x)\} \\ E(\Gamma, M\,N, \sigma) & = & E(\Gamma, M, \alpha \to \sigma) \cup E(\Gamma, N, \alpha) \\ E(\Gamma, \lambda x.\,M, \sigma) & = & E(\Gamma \cup \{x{:}\alpha\}, M, \beta) \cup \{\alpha \to \beta \sim \sigma\} \end{array}$$

- 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, σ)

$$\begin{array}{lcl} E(x{:}\gamma \to \delta, x, \alpha \to \beta \to \alpha) & = & ??? \\ E(x{:}\gamma \to \delta, x\,x, \alpha \to \beta \to \alpha) & = & ??? \\ E(x{:}\gamma \to \delta, \lambda x.\,x, \alpha \to \beta \to \alpha) & = & ??? \end{array}$$

Главная пара (Principle Pair)

Определение

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

- $\Gamma \vdash M : \sigma$
- $\Gamma' \vdash M : \sigma' \ \Rightarrow \ \exists S \ \left[S(\Gamma) \subseteq \Gamma' \ \wedge \ S(\sigma) \equiv \sigma' \right]$

Пример

Для $M = \lambda x.\, x\, y$ имеем

$$PP(M) = (y : \alpha, (\alpha \to \beta) \to \beta)$$

$$y : \alpha \vdash (\lambda x.\, x\, y) : (\alpha \to \beta) \to \beta$$

Теорема Хиндли – Милнера

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

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

Пусть $FV(M) = \{x_1, \ldots, x_n\}$, $\Gamma_0 = \{x_1 : \alpha_1, \ldots, x_n : \alpha_n\}$ и $\sigma_0 = \beta$.

Алгоритм PP

$$
\begin{array}{lcl}
PP(M) \mid U(E(\Gamma_0, M, \sigma_0)) \equiv \text{ошибка} & = & \text{ошибка} \\
PP(M) \mid U(E(\Gamma_0, M, \sigma_0)) \equiv S & = & (S(\Gamma_0), S(\sigma_0))
\end{array}
$$

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

Главный тип (Principle Type)

Определение

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

- $\vdash M : \sigma$
- $\vdash M : \sigma' \;\Rightarrow\; \exists S \;\big[S(\sigma) \equiv \sigma'\big]$

Следствие теоремы Хиндли – Милнера

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

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

План лекции

Где HM не справляется?

Есть одно тонкое место, в котором алгоритм HM не работает:

```
GHCi>Prelude> :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 HM не справляется?

Есть одно тонкое место, в котором алгоритм HM не работает:

```
GHCi>Prelude> :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
```

`let`-полиморфизм

- Для (частичного) снятия этой проблемы нужен более точный контроль за местом применения функции.
- Это делают с помощью `let`, рассматривая его как примитив (для вывода типов), 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 этом случае тип функции `f` неявно трактуется как `f :: forall a. a -> a` и снятие квантора происходит в момент аппликации.

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 включить расширение и явно указать тип, поскольку вывод типов для систем высших рангов (> 2) неразрешим.

```
GHCi> :set -XRank2Types
GHCi> let { f :: (forall a. a -> a) -> (Char,Bool);
            f g = (g 'z',g True) } in f id
('z',True)
```

План лекции

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 2 :@

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 $(\lambda x.\lambda y.\, x)\,(\lambda z.\, z)$.

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) = freeVars t1 `union` freeVars t2
freeVars (Lam v t)  = freeVars t \\ [v]
```

Определение типа

- Типы можно закодировать так

```
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)
```

- Полезными могут оказаться пустой контекст и функция расширения контекста

```
emptyEnv :: Env
emptyEnv = Env []

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

Литература

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.