Вывод типов и полиморфизм

Роман Чепляка

30 мая 2009
LtU@Kiev

Вывод типов по Карри и Чёрчу

По Карри	По Чёрчу
Терм не содержит информации о типах	Терм содержит информацию о типах
Вывод типа — определение типа по терму	Вывод типа — восстановление терма по нетипизированному претерму
In: $\lambda x.\lambda y.xy$ (терм) Out: $(\alpha \to \beta) \to \alpha \to \beta$ (тип)	In: $\lambda x.\lambda y.xy$ (претерм) Out: $\lambda x{:}(\alpha \to \beta).\lambda y{:}\alpha.xy$ (терм)

```
Prelude> :t \x y -> x y
\x y -> x y :: (a -> b) -> a -> b
```

Синтаксис $\lambda^{\to Curry}$

Типы:

$$T ::= b \mid x \mid T \to T$$

Правила вывода:

$$\Gamma, x : \sigma \vdash x : \sigma$$

$$\frac{\Gamma \vdash M : (\sigma \to \tau) \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash (MN : \tau)}$$

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

Пример вывода типа

Пример. Для любых типов a, b можно вывести
$\vdash (\lambda x.\lambda y.xy) : (a \to b) \to a \to b$:

$$\dfrac{\dfrac{\dfrac{x : (a \to b) \vdash x : (a \to b) \qquad y : a \vdash y : a}{x : (a \to b), y : a \vdash xy : b}}{x : (a \to b) \vdash (\lambda y.xy) : (a \to b)}}{\vdash \lambda x.\lambda y.xy : (a \to b) \to a \to b}$$

В частности:
$\vdash (\lambda x.\lambda y.xy) : (Integer \to Bool) \to Integer \to Bool$
$\vdash (\lambda x.\lambda y.xy) : (String \to String) \to String \to String$

$\Rightarrow$ задача вывода типа в $\lambda^{\to}$ не имеет однозначного решения, необходимо найти все возможные типы.

λ-исчисление со схемами типов

Расширим множество типов $\lambda^{\rightarrow}$ переменными типов:

$$T ::= b \mid x \mid \alpha \mid T \rightarrow T$$

Полученное λ-исчисление назовем $\lambda_t^{\rightarrow}$.

Тогда $\vdash_{\lambda_t^{\rightarrow}} (\lambda x.\lambda y.xy) : (\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \beta$.

Подстановки и примеры

Подстановка типов — отображение

$$S\colon \mathit{Types} \to \mathit{Types},$$

такое что

$$S(\sigma \to \tau) = S(\sigma) \to S(\tau),$$

$$S(b) = b.$$

Пример. $S = [\alpha := \mathit{Integer}, \beta := \mathit{Bool}] \Rightarrow$

$$S((\alpha \to \beta) \to \alpha \to \beta) = (\mathit{Integer} \to \mathit{Bool}) \to \mathit{Integer} \to \mathit{Bool}$$

Если $S(\sigma) = \tau$, тип τ называется *примером* типа σ.
Пусть $\vdash M : \sigma$. Тип σ называется *главным* типом M, если любой другой тип M является примером σ.

Составление уравнений

$E(\Gamma, M, \sigma)$ — набор уравнений.

$$E(\Gamma, x, \sigma) = \{\sigma = \Gamma(x)\}$$

$$E(\Gamma, MN, \sigma) = E(\Gamma, M, \alpha \to \sigma) \cup E(\Gamma, N, \alpha)$$

$$E(\Gamma, \lambda x.M, \sigma) = E(\Gamma \cup \{x : \alpha\}, M, \beta) \cup \{\sigma = \alpha \to \beta\}$$

Подстановка типов S *унифицирует* уравнение $\sigma = \tau$, если

$$S(\sigma) = S(\tau).$$

Алгоритм унификации системы уравнений E

$Unify(\emptyset) = id$

$Unify(E \cup \{b_1 = b_2\}) =$
$\quad$ **if** $b_1 = b_2$
$\quad\quad$ **then** $Unify(E)$
$\quad\quad$ **else** $fail$

$Unify(E \cup \{b = \sigma \to \tau\}) = fail$

$Unify(E \cup \{\alpha = \tau\}) =$
$\quad$ **if** $\tau = \alpha$ **then** $Unify(E)$
$\quad\quad$ **else if** α *входит в* τ **then** $fail$
$\quad\quad\quad$ **else** $Unify([\alpha := \tau]E) \circ [\alpha := \tau]$

$Unify(E \cup \{\sigma_1 \to \sigma_2 = \tau_1 \to \tau_2\}) = Unify(E \cup \{\sigma_1 = \tau_1, \sigma_2 = \tau_2\})$

Свойства алгоритма унификации

Алгоритм *Unify* находит унифицирующую подстановку, если она существует, и завершается неудачей (*fail*) в противном случае.

Алгоритм *Unify* находит наиболее общую подстановку: если T унифицирует E, $S = Unify(E)$, то

$$\exists R : T = R \circ S$$

Следовательно, если $\vdash M : \sigma$, то M имеет главный тип.

Немного истории

- Карри использовал НМ неформально в 1950х, возможно даже в 1930х, перед тем как описал алгоритм формально в 1967 г. (опубликовано в 1969 г.). Карри также описал алгоритм унификации.
- Алгоритм Хиндли (1967) опирается на алгоритм унификации Робинзона.
- Алгоритм Милнера также опирается на алгоритм Робинзона.
- Дж. Моррис описал алгоритм решения уравнений в его диссертации в 1968 г, включая алгоритм унификации.
- К. Мередит использовал подобный алгоритм в 1950х, работая над пропозициональной логикой.
- Тарский предположительно использовал алгоритм унификации в своих ранних работах (1920е).

Мораль: вероятно, кому-то не помешало бы научиться читать. Или кому-то другому — писать. (*Роджер Хиндли*)

Примеры

Выведем тип $\lambda x.\lambda y.xy$:

$$
\begin{aligned}
&E(\emptyset, \lambda x.\lambda y.xy, \sigma) = \\
&\quad E(\{x:\gamma\}, \lambda y.xy, \delta) \cup \{\sigma = \gamma \to \delta\} = \\
&\quad E(\{x:\gamma, y:\alpha\}, xy, \beta) \cup \{\sigma = \gamma \to \delta, \delta = \alpha \to \beta\} = \\
&\quad E(\{x:\gamma, y:\alpha\}, x, \nu \to \beta) \cup E(\{x:\gamma, y:\alpha\}, y, \nu) \\
&\quad \cup \{\sigma = \gamma \to \delta, \delta = \alpha \to \beta\} = \\
&\quad \{\nu \to \beta = \gamma, \nu = \alpha, \sigma = \gamma \to \delta, \delta = \alpha \to \beta\}
\end{aligned}
$$

Примеры

$$S = Unify(\{\nu \to \beta = \gamma, \nu = \alpha, \sigma = \gamma \to \delta, \delta = \alpha \to \beta\}) =$$
$$Unify(\{\nu = \alpha, \sigma = (\nu \to \beta) \to \delta, \delta = \alpha \to \beta\}) \circ [\gamma := \nu \to \beta] =$$
$$Unify(\{\sigma = (\alpha \to \beta) \to \delta, \delta = \alpha \to \beta\}) \circ [\nu := \alpha] \circ [\gamma := \nu \to \beta] =$$
$$Unify(\{\delta = \alpha \to \beta\}) \circ [\sigma := (\alpha \to \beta) \to \delta] \circ [\nu := \alpha] \circ [\gamma := \nu \to \beta] =$$
$$[\delta := \alpha \to \beta] \circ [\sigma := (\alpha \to \beta) \to \delta] \circ [\nu := \alpha] \circ [\gamma := \nu \to \beta]$$

$$PT(\lambda x.\lambda y.xy) = S(\sigma) = (\alpha \to \beta) \to \alpha \to \beta$$

Примеры

Терм $\omega = \lambda x.xx$ не типизируем в $\lambda_t^{\rightarrow}$, так как в $\lambda_t^{\rightarrow}$ каждый терм имеет нормальную форму, а $\Omega = \omega\omega$ ее не имеет ($\Omega \rightarrow_\beta \Omega$).

$$E(\emptyset, \lambda x.xx, \sigma) = E(\{x : \alpha\}, xx, \beta) \cup \{\sigma = \alpha \rightarrow \beta\} =$$
$$E(\{x : \alpha\}, x, \gamma \rightarrow \beta) \cup E(\{x : \alpha\}, x, \gamma) \cup \{\sigma = \alpha \rightarrow \beta\} =$$
$$\{\alpha = \gamma \rightarrow \beta, \alpha = \gamma, \sigma = \alpha \rightarrow \beta\}$$

$$\mathit{Unify}(\{\alpha = \gamma \rightarrow \beta, \alpha = \gamma, \sigma = \alpha \rightarrow \beta\}) =$$
$$\mathit{Unify}(\{\gamma \rightarrow \beta = \gamma, \sigma = (\gamma \rightarrow \beta) \rightarrow \beta\}) \circ [\alpha := \gamma \rightarrow \beta] = \mathit{fail}$$

```
Prelude> :t \x -> x x

<interactive>:1:6:
    Occurs check: cannot construct the infinite type: t = t -> t1
    Probable cause: `x' is applied to too many arguments
    In the expression: x x
    In the expression: \ x -> x x
```

Ограничения HM

$$\textbf{let } x = M \textbf{ in } N \stackrel{\text{def}}{=} (\lambda x.N)M$$

$$\begin{aligned}
A = \textbf{let } & \mathit{const} = \lambda x.\lambda y.x \\
& \mathit{id} = \lambda x.x \\
& \textbf{in } \mathit{const}\ \mathit{id}
\end{aligned}$$

$$PT(A) = \alpha \to \beta \to \beta$$

$$\begin{aligned}
B = \textbf{let } & \mathit{id} = \lambda x.x \\
& \textbf{in } \mathit{id}\ \mathit{id}
\end{aligned}$$

$$B = (\lambda \mathit{id}.\mathit{id}\ \mathit{id})(\lambda x.x) = (\lambda x.xx)(\lambda x.x)$$
$$PT(B) = \mathit{fail}$$

Полиморфизм по-эмелевски

$$\frac{\Gamma \vdash U : \sigma \qquad \Gamma \vdash V[x := U] : \tau}{\Gamma \vdash \textbf{let } x = U \textbf{ in } V : \tau} \text{ (let)}$$

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

$$\frac{\vdash \lambda x.x : \alpha \to \alpha \qquad \vdash (id\ id)[id := \lambda x.x] : \alpha \to \alpha}{\vdash \textbf{let } id = \lambda x.x \textbf{ in } id\ id : \alpha \to \alpha}$$

$\lambda_t^{\to}$, расширенное правилом (let), называется λ-исчислением с let-полиморфизмом или ML-полиморфизмом.

Когда и этого мало

```
data Monad m = Mon { return :: a -> m a,
                     bind   :: m a -> (a -> m b) -> m b }

mapM :: Monad m -> (a -> m b) -> [a] -> m [b]
mapM m@(Mon { return = ret, bind = bnd }) f xs
  = case xs of
      []     -> ret []
      (x:xs) -> f x           `bnd` \y  ->
                mapM m f xs   `bnd` \ys ->
                ret (y:ys)
```

Решение — полноценное полиморфное λ-исчисление (System F и др.). Вывод типов алгоритмически неразрешим для System F и большинства ее подсистем.