Просто типизированное лямбда-исчисление

Дмитрий Халанский

21 сентября 2020 г.

Контекст типизации: пример на Си

```
void arbitrary(int a, char b, char *i)
{
        // point 0
        int c = 5;
        for (int i = 0; i < 15; ++i) {
                i -= 1; // point 1
                long r = i;
                i *= 0; // point 2
        }
        c = a + (int)b; // point 3
}
```

Контекст типизации в каждой из точек (не считая глобальных определений):

0. $\{a : \text{int}, b : \text{char}, i : \text{char}*\}$

Контекст типизации: пример на Си

```
void arbitrary(int a, char b, char *i)
{
        // point 0
        int c = 5;
        for (int i = 0; i < 15; ++i) {
                i -= 1; // point 1
                long r = i;
                i *= 0; // point 2
        }
        c = a + (int)b; // point 3
}
```

Контекст типизации в каждой из точек (не считая глобальных определений):

0. $\{a : \text{int}, b : \text{char}, i : \text{char}*\}$
1. $\{a : \text{int}, b : \text{char}, c : \text{int}, i : \text{int}\}$

Контекст типизации: пример на Си

```
void arbitrary(int a, char b, char *i)
{
        // point 0
        int c = 5;
        for (int i = 0; i < 15; ++i) {
                i -= 1; // point 1
                long r = i;
                i *= 0; // point 2
        }
        c = a + (int)b; // point 3
}
```

Контекст типизации в каждой из точек (не считая глобальных определений):

0. $\{a : \text{int}, b : \text{char}, i : \text{char}*\}$
1. $\{a : \text{int}, b : \text{char}, c : \text{int}, i : \text{int}\}$
2. $\{a : \text{int}, b : \text{char}, c : \text{int}, i : \text{int}, r : \text{long}\}$

Контекст типизации: пример на Си

```
void arbitrary(int a, char b, char *i)
{
        // point 0
        int c = 5;
        for (int i = 0; i < 15; ++i) {
                i -= 1; // point 1
                long r = i;
                i *= 0; // point 2
        }
        c = a + (int)b; // point 3
}
```

Контекст типизации в каждой из точек (не считая глобальных определений):

0. $\{a : \text{int}, b : \text{char}, i : \text{char}*\}$
1. $\{a : \text{int}, b : \text{char}, c : \text{int}, i : \text{int}\}$
2. $\{a : \text{int}, b : \text{char}, c : \text{int}, i : \text{int}, r : \text{long}\}$
3. $\{a : \text{int}, b : \text{char}, c : \text{int}, i : \text{char}*\}$

Что означают типы

Найдите все возможные варианты для терма M в каждом из случаев:

- $\{x : \beta\} \vdash M : \alpha \to \alpha$
- $\{x : \beta\} \vdash M : \alpha \to \beta$
- $\{x : \beta\} \vdash M : \alpha \to \gamma$

Что означают типы

Найдите все возможные варианты для терма M в каждом из случаев:

- $\{x : \beta\} \vdash M : \alpha \to \alpha$
- $\{x : \beta\} \vdash M : \alpha \to \beta$
- $\{x : \beta\} \vdash M : \alpha \to \gamma$

Мораль: тип даёт гарантию, что если в M подать аргументы заданного *вызывающим* вида, то M обязательно вернёт результат заданного *вызывающим* вида; равенство переменных и структура стрелок в типе накладывают ограничения на вызывающего.

- В первом случае вызывающий обязан ожидать, что M вернёт значение того же типа, что и подаётся на вход; $M = \lambda y.\, y$.
- Во втором — вызывающий может подать на вход что угодно, но ожидать он обязан значение того же типа, что и x; $M = \lambda y.\, x$.
- В третьем — вызывающий имеет право подать что угодно и требовать что угодно. Таких M не может быть.

Деревья вывода

- Деревья вывода — это конструкции, полученные применением *конечного количества* заранее заданных правил вывода.
- По правилам вывода иногда очевиден алгоритм построения соответствующих деревьев, но это не обязательно.
- В деревья вывода не вшит смысл, это *синтаксические*, машинно проверяемые конструкции. Смысл деревья обретают, если доказать теоремы вроде “если существует дерево вывода, в корне которого написано X, то выполняется Y”.

Правила вывода для фактов типизации (напоминание)

1. $$\frac{(x : \sigma) \in \Gamma}{\Gamma \vdash x : \sigma}\ 1$$

2. $$\frac{\Gamma \vdash A : \sigma \to \tau \quad \Gamma \vdash B : \sigma}{\Gamma \vdash A\ B : \tau}\ 2$$

3. $$\frac{\{x : \sigma\} \cup \Gamma \vdash M : \tau}{\Gamma \vdash \lambda x^{\sigma}.\, M : \sigma \to \tau}\ 3$$

К счастью для нас, для любого высказывания вида $\gamma \vdash M : \tau$, где M заранее задан, существует только одно правило (в зависимости от вида M), в заключении которого могло бы оказаться это суждение. Это значит, что по правилам очевиден алгоритм вывода типов: если M — переменная, попытаться применить 1; если это аппликация, попытаться применить 2; иначе 3.

Типизация комбинатора B (0)

Рассмотрим, как, пытаясь применить эти три правила, можно определить тип терма.
Терм замкнутый, так что заранее знаем, что достаточно пустого контекста.

$$\emptyset \vdash \lambda f\ g\ x.\ f\ (g\ x) :?_1 \to ?_2$$

Типизация комбинатора B (1)

$\lambda f\ g\ x.\ f\ (g\ x)$ — это абстракция. Применяем правило 3.

$$\frac{\{f :?_1\} \vdash \lambda g\ x.\ f\ (g\ x) :?_2}{\emptyset \vdash \lambda f\ g\ x.\ f\ (g\ x) :?_1 \to ?_2}\ 3$$

Типизация комбинатора B (2)

$\lambda g\ x.\ f\ (g\ x)$ — это абстракция. Применяем правило 3. Заметим, что $?_2$ — это тип-стрелка, и разобьём его: $?_2 = ?_3 \to ?_4$.

$$\dfrac{\dfrac{\{f :?_1, g :?_3\} \vdash \lambda x.\ f\ (g\ x) :?_4}{\{f :?_1\} \vdash \lambda g\ x.\ f\ (g\ x) :?_3 \to ?_4}\ 3}{\emptyset \vdash \lambda f\ g\ x.\ f\ (g\ x) :?_1 \to ?_3 \to ?_4}\ 3$$

Типизация комбинатора B (3)

$\lambda x.\ f\ (g\ x)$ — это снова абстракция (но последняя). Снова правило 3. $?_4$ — это тип-стрелка, пусть $?_4 = ?_5 \to ?_6$.

$$\dfrac{\dfrac{\dfrac{\{f :?_1, g :?_3, x :?_5\} \vdash f\ (g\ x) :?_6}{\{f :?_1, g :?_3\} \vdash \lambda x.\ f\ (g\ x) :?_5 \to ?_6}\ 3}{\{f :?_1\} \vdash \lambda g\ x.\ f\ (g\ x) :?_3 \to ?_5 \to ?_6}\ 3}{\emptyset \vdash \lambda f\ g\ x.\ f\ (g\ x) :?_1 \to ?_3 \to ?_5 \to ?_6}\ 3$$

Типизация комбинатора B (3)

$f\ (g\ x)$ — это, наконец, аппликация. Применимо правило 2.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\{f :?_1, g :?_3, x :?_5\} \vdash f :?_7 \to ?_6 \quad \{f :?_1, g :?_3, x :?_5\} \vdash gx :?_7}{\{f :?_1, g :?_3, x :?_5\} \vdash f\ (g\ x)}\ 2}{\{f :?_1, g :?_3\} \vdash \lambda x.\ f\ (g\ x) :?_5 \to ?_6}\ 3}{\{f :?_1\} \vdash \lambda g\ x.\ f\ (g\ x) :?_3 \to ?_5 \to ?_6}\ 3}{\emptyset \vdash \lambda f\ g\ x.\ f\ (g\ x) :?_1 \to ?_3 \to ?_5 \to ?_6}\ 3$$

Типизация комбинатора B (4)

Пойдём в левую ветку. Там применимо правило типизации 1. Значит, $?_1 = ?_7 \to ?_6$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\{f : ?_7 \to ?_6, g : ?_3, x : ?_5\} \vdash f : ?_7 \to ?_6}\,1 \quad \{f : ?_7 \to ?_6, g : ?_3, x : ?_5\} \vdash gx : ?_7}{\{f : ?_7 \to ?_6, g : ?_3, x : ?_5\} \vdash f\ (g\ x)}\,2}{\{f : ?_7 \to ?_6, g : ?_3\} \vdash \lambda x.\ f\ (g\ x) : ?_5 \to ?_6}\,3}{\{f : ?_7 \to ?_6\} \vdash \lambda g\ x.\ f\ (g\ x) : ?_3 \to ?_5 \to ?_6}\,3}{\emptyset \vdash \lambda f\ g\ x.\ f\ (g\ x) : (?_7 \to ?_6) \to ?_3 \to ?_5 \to ?_6}\,3$$

Типизация комбинатора B (5)

Теперь пойдём в правую ветку. Там аппликация и правило типизации 2.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\{f : ?_7 \to ?_6, g : ?_3, x : ?_5\} \vdash f : ?_7 \to ?_6}\,1 \quad \dfrac{\{f : ?_7 \to ?_6, g : ?_3, x : ?_5\} \vdash g : ?_8 \to ?_7 \quad \{f : ?_7 \to ?_6, g : ?_3, x : ?_5\} \vdash x : ?_8}{\{f : ?_7 \to ?_6, g : ?_3, x : ?_5\} \vdash gx : ?_7}\,2}{\{f : ?_7 \to ?_6, g : ?_3, x : ?_5\} \vdash f\ (g\ x)}\,2}{\{f : ?_7 \to ?_6, g : ?_3\} \vdash \lambda x.\ f\ (g\ x) : ?_5 \to ?_6}\,3}{\{f : ?_7 \to ?_6\} \vdash \lambda g\ x.\ f\ (g\ x) : ?_3 \to ?_5 \to ?_6}\,3}{\emptyset \vdash \lambda f\ g\ x.\ f\ (g\ x) : (?_7 \to ?_6) \to ?_3 \to ?_5 \to ?_6}\,3$$

Типизация комбинатора B (6)

Пойдём в левую ветку. Там применимо правило 1, тогда $?_3 = ?_8 \to ?_7$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\{f : ?_7 \to ?_6, g : ?_8 \to ?_7, x : ?_5\} \vdash f : ?_7 \to ?_6}\,1 \quad \dfrac{\dfrac{}{\{f : ?_7 \to ?_6, g : ?_8 \to ?_7, x : ?_5\} \vdash g : ?_8 \to ?_7}\,1 \quad \{f : ?_7 \to ?_6, g : ?_8 \to ?_7, x : ?_5\} \vdash x : ?_8}{\{f : ?_7 \to ?_6, g : ?_8 \to ?_7, x : ?_5\} \vdash gx : ?_7}\,2}{\{f : ?_7 \to ?_6, g : ?_8 \to ?_7, x : ?_5\} \vdash f\ (g\ x)}\,2}{\{f : ?_7 \to ?_6, g : ?_8 \to ?_7\} \vdash \lambda x.\ f\ (g\ x) : ?_5 \to ?_6}\,3}{\{f : ?_7 \to ?_6\} \vdash \lambda g\ x.\ f\ (g\ x) : (?_8 \to ?_7) \to ?_5 \to ?_6}\,3}{\emptyset \vdash \lambda f\ g\ x.\ f\ (g\ x) : (?_7 \to ?_6) \to (?_8 \to ?_7) \to ?_5 \to ?_6}\,3$$

Типизация комбинатора B (7)

Пойдём в правую ветку и получим итоговое дерево путём применения правила 1 и замечания, что тогда $?_8 = ?_5$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\{f : ?_7 \to ?_6, g : ?_5 \to ?_7, x : ?_5\} \vdash f : ?_7 \to ?_6}\,1 \quad \dfrac{\dfrac{\dfrac{}{\{f : ?_7 \to ?_6, g : ?_5 \to ?_7, x : ?_5\} \vdash g : ?_5 \to ?_7}\,1 \quad \dfrac{}{\{f : ?_7 \to ?_6, g : ?_5 \to ?_7, x : ?_5\} \vdash x : ?_5}\,1}{\{f : ?_7 \to ?_6, g : ?_5 \to ?_7, x : ?_5\} \vdash gx : ?_7}\,2}{\{f : ?_7 \to ?_6, g : ?_5 \to ?_7, x : ?_5\} \vdash f\ (g\ x)}\,2}{\{f : ?_7 \to ?_6, g : ?_5 \to ?_7\} \vdash \lambda x.\ f\ (g\ x) : ?_5 \to ?_6}\,3}{\{f : ?_7 \to ?_6\} \vdash \lambda g\ x.\ f\ (g\ x) : (?_5 \to ?_7) \to ?_5 \to ?_6}\,3}{\emptyset \vdash \lambda f\ g\ x.\ f\ (g\ x) : (?_7 \to ?_6) \to (?_5 \to ?_7) \to ?_5 \to ?_6}\,3$$

Полученный тип — $(?_7 \to ?_6) \to (?_5 \to ?_7) \to ?_5 \to ?_6$ — совпадает с интуицией.

Не всё хорошее типизируется

Хотелось бы, чтобы все хорошие программы типизировались, а все плохие — нет. Это невозможно (проблема останова, теорема Успенского-Райса). Всё, что типизируется, хорошее, но обратное не всегда верно.

$$2\ \mathsf{K}$$

Этот терм сильно нормализуем и редуцируется так:

$$2\ \mathsf{K} = \lambda z.\, \mathsf{K}\ (\mathsf{K}\ z) = \lambda z\ a\ b.\, z$$

Однако протипизировать это не получится.

- Наиболее общий тип K — это

Не всё хорошее типизируется

Хотелось бы, чтобы все хорошие программы типизировались, а все плохие — нет. Это невозможно (проблема останова, теорема Успенского-Райса). Всё, что типизируется, хорошее, но обратное не всегда верно.

$$2\ \mathsf{K}$$

Этот терм сильно нормализуем и редуцируется так:

$$2\ \mathsf{K} = \lambda z.\, \mathsf{K}\ (\mathsf{K}\ z) = \lambda z\ a\ b.\, z$$

Однако протипизировать это не получится.

- Наиболее общий тип K — это $\gamma \to \tau \to \gamma$.
- $2 := \lambda s\ z.\, s\ (s\ z)$. Наиболее общий тип 2 — это

Не всё хорошее типизируется

Хотелось бы, чтобы все хорошие программы типизировались, а все плохие — нет. Это невозможно (проблема останова, теорема Успенского-Райса). Всё, что типизируется, хорошее, но обратное не всегда верно.

2 K

Этот терм сильно нормализуем и редуцируется так:

$$2\ \mathsf{K} = \lambda z.\, \mathsf{K}\ (\mathsf{K}\ z) = \lambda z\ a\ b.\, z$$

Однако протипизировать это не получится.

- Наиболее общий тип K — это $\gamma \to \tau \to \gamma$.
- $2 := \lambda s\ z.\, s\ (s\ z)$. Наиболее общий тип 2 — это $(\alpha \to \alpha) \to \alpha \to \alpha$.
- По правилу типизации аппликации, применение 2 к K накладывает на γ, τ, α ограничения

Не всё хорошее типизируется

Хотелось бы, чтобы все хорошие программы типизировались, а все плохие — нет. Это невозможно (проблема останова, теорема Успенского-Райса). Всё, что типизируется, хорошее, но обратное не всегда верно.

$$2\ \mathsf{K}$$

Этот терм сильно нормализуем и редуцируется так:

$$2\ \mathsf{K} = \lambda z.\, \mathsf{K}\ (\mathsf{K}\ z) = \lambda z\ a\ b.\, z$$

Однако протипизировать это не получится.

- Наиболее общий тип K — это $\gamma \to \tau \to \gamma$.
- $2 := \lambda s\ z.\, s\ (s\ z)$. Наиболее общий тип 2 — это $(\alpha \to \alpha) \to \alpha \to \alpha$.
- По правилу типизации аппликации, применение 2 к K накладывает на γ, τ, α ограничения $\alpha = \gamma, \alpha = \tau \to \gamma$. Противоречие.

β-эквивалентность не сохраняет тип

Из лекции известно, что если M редуцируется к N и $\Gamma \vdash M : \tau$, то $\Gamma \vdash N : \tau$.
Неправда, что если M редуцируется к N и $\Gamma \vdash N : \tau$, то $\Gamma \vdash M : \tau$.
Пример: $N = \lambda x.\, x$, $N : \alpha \to \alpha$; $M = \lambda x.\, (\lambda y.\, x)\ (x\ \mathsf{I})$, и за одно редуцирование M переходит в N, но $M : \alpha \to \alpha$ не выполняется: тогда бы подтерм $x\ \mathsf{I}$ не типизировался, так как α не является стрелочным типом.

η-эквивалентность не сохраняет тип

Если $\lambda x.\, M\ x : \tau$ (где x не является свободным в M), то $M : \tau$.
Если $M : \tau$, то не обязательно $\lambda x.\, M\ x : \tau$. Пример: $\mathsf{I} : \alpha \to \alpha$, но $\lambda x\ y.\, x\ y : (\beta \to \gamma) \to (\beta \to \gamma)$ нельзя приписать тип $\alpha \to \alpha$: тогда бы подтерм $x\ y$ не типизировался, так как α не является стрелочным типом.