

Типы в языках программирования

Лекция 3. Простые расширения

Денис Николаевич Москвин

СПбАУ РАН

01.03.2018


- 1 Соответствие Карри-Говарда
- 2 Простейшие расширения
- 3 Простые расширения
- 4 Рекурсия общего вида

- 1 Соответствие Карри-Говарда
- 2 Простейшие расширения
- 3 Простые расширения
- 4 Рекурсия общего вида

Типизация

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \quad T - \text{Var}$$
$$\frac{\Gamma, x : T \vdash t : S}{\Gamma \vdash \lambda x : T. t : T \rightarrow S} \quad T - \text{Abs}$$
$$\frac{\Gamma \vdash t_1 : T \rightarrow S \quad \Gamma \vdash t_2 : T}{\Gamma \vdash t_1 t_2 : S} \quad T - \text{App}$$

- Сравним дерево вывода типа для $\lambda x.u$. x и доказательство $P \rightarrow Q \rightarrow P^1$.
- Правила введения и удаления полезно осознавать для типов, с которыми мы имеем дело. Каковы они для `Bool`? `Nat`?

¹ P и Q можно рассматривать как неинтерпретируемые базовые типы. 

- Взгляд на правила типизации, как на правила введения и удаления, дает связь с конструктивной логикой:

ЛОГИКА	ТЕОРИЯ ТИПОВ
утверждение	тип
импликация $P \rightarrow Q$	тип функции $P \rightarrow Q$
доказательство утверждения P	терм t типа P
утверждения P доказуемо	тип P обитаем

- Вычисление на термах соответствует правилу устранения сечений, то есть упрощению доказательств.
- Соответствие Карри-Говарда имеет место не только для импликационного фрагмента пропозициональной логики.

- 1 Соответствие Карри-Говарда
- 2 Простейшие расширения**
- 3 Простые расширения
- 4 Рекурсия общего вида

Новые синтаксические формы

```
t ::= ...  
    unit  
v ::= ...  
    unit  
T ::= ...  
    Unit
```

Новые правила типизации

$$\Gamma \vdash \text{unit} : \text{Unit} \quad (T - \text{Unit})$$

- `unit` — единственный возможный результат вычисления выражения типа `Unit`.
- В рамках соответствия Карри-Говарда `Unit` рассматривается как заведомо истинное утверждение.

Можно ввести *синтаксический сахар* (производные формы):

Новые производные формы

$$t_1; t_2 \equiv (\lambda x : \text{Unit}. t_2) t_1 \\ x \notin \text{FV}(t_2)$$

- Оператор *последовательного исполнения* (sequencing notation) используется для описания последовательных вычислений при наличии встроенных эффектов.
- Семантика должна быть, конечно, **энергичной**.
- Чтобы не делать оговорку $x \notin \text{FV}(t_2)$ для неиспользуемой переменной x , вводят еще одну производную форму — *связывание-пустышку*: $\lambda_ : \text{Unit}. t_2$.

Последовательные вычисления можно было бы ввести и через примитивы

Возможные правила вычисления

$$\frac{t_1 \longrightarrow t'_1}{t_1; t_2 \longrightarrow t'_1; t_2} \quad (\text{E - Seq})$$

$$v; t_2 \longrightarrow t_2 \quad (\text{E - SeqNext})$$

Возможные правила типизации

$$\frac{\Gamma \vdash t_1 : \text{Unit} \quad \Gamma \vdash t_2 : T}{\Gamma \vdash t_1; t_2 : T} \quad (\text{T - Seq})$$

Несложно доказать эквивалентность: и вычисление, и типизацию можно менять местами с раскрытием сокращения.

Новые синтаксические формы

$$T ::= \dots$$

Bot

- Тип Bot не населен: нет выражений, имеющих такой тип.
- В рамках соответствия Карри-Говарда Bot рассматривается как заведомо ложное утверждение.
- В конструктивных логиках естественно определять отрицание через Bot:

Новые производные формы

$$\neg T \equiv T \rightarrow \text{Bot}$$

Новые синтаксические формы

$$t ::= \dots$$
$$t \text{ as } T$$

Новые правила вычисления и типизации

$$v \text{ as } T \longrightarrow v \quad (\text{E - Ascribe})$$

$$\frac{t \longrightarrow t'}{t \text{ as } T \longrightarrow t' \text{ as } T} \quad (\text{E - Ascribe1})$$

$$\frac{\Gamma \vdash t : T}{\Gamma \vdash t \text{ as } T : T} \quad (\text{T - Ascribe})$$

- Служит для целей документации, обслуживания механизма синонимов типов и конкретизации в системах, в которых у термина может быть несколько типов.

- Можно ли реализовать явное присисывание типов, как синтаксический сахар?

- Можно ли реализовать явное приписывание типов, как синтаксический сахар? Да:

Новые производные формы

$$t \text{ as } T \equiv (\lambda x : T. x)t$$

- При этом вычислительные правила (E – Ascribe) и (E – Ascribe1) выполняются (для энергичной семантики).
- А можно ли «засахарить» «энергичное» стирание:

Альтернативное правило вычисления

$$\frac{t \longrightarrow t'}{t \text{ as } T \longrightarrow t'} \quad (\text{E – Ascribe1A})$$

- **Самостоятельно.**

Новые синтаксические формы

$$t ::= \dots$$

$$\text{let } x = t \text{ in } t$$

Новые правила вычисления и типизации

$$\text{let } x = v \text{ in } t \longrightarrow [x \mapsto v]t \quad (\text{E - LetV})$$

$$\frac{t_1 \longrightarrow t'_1}{\text{let } x = t_1 \text{ in } t_2 \longrightarrow \text{let } x = t'_1 \text{ in } t_2} \quad (\text{E - Let})$$

$$\frac{\Gamma \vdash t_1 : T \quad \Gamma, x : T \vdash t_2 : S}{\Gamma \vdash \text{let } x = t_1 \text{ in } t_2 : S} \quad (\text{T - Let})$$

- Связывание `let` можно определить как производную форму:

$$\text{let } x = t_1 \text{ in } t_2 \equiv (\lambda x : T_1. t_2)t_1$$

- Однако это потребует реконструкции типа при трансляции.
- В простых полиморфных системах это позволяет несколько увеличить «мощность» полиморфизма, поэтому `let` часто вводят как примитив.
- А почему бы не ввести вычислительное правило:
 $\text{let } x = t_1 \text{ in } t_2 \longrightarrow [x \mapsto t_1]t_2?$

- 1 Соответствие Карри-Говарда
- 2 Простейшие расширения
- 3 Простые расширения**
- 4 Рекурсия общего вида

Новые синтаксические формы

```
t ::= ...  
    {t,t}  
    t.1  
    t.2  
v ::= ...  
    {v,v}  
T ::= ...  
    T × T
```

Новые правила вычисления

$$\{v_1, v_2\}.1 \longrightarrow v_1 \quad (\text{E - PairBeta1})$$

$$\{v_1, v_2\}.2 \longrightarrow v_2 \quad (\text{E - PairBeta2})$$

$$\frac{t \longrightarrow t'}{t.1 \longrightarrow t'.1} \quad (\text{E - Proj1})$$

$$\frac{t \longrightarrow t'}{t.2 \longrightarrow t'.2} \quad (\text{E - Proj2})$$

$$\frac{t_1 \longrightarrow t'_1}{\{t_1, t_2\} \longrightarrow \{t'_1, t_2\}} \quad (\text{E - Pair1})$$

$$\frac{t_2 \longrightarrow t'_2}{\{v_1, t_2\} \longrightarrow \{v_1, t'_2\}} \quad (\text{E - Pair2})$$

Новые правила типизации

$$\frac{\Gamma \vdash t : T \quad \Gamma \vdash s : S}{\Gamma \vdash \{t, s\} : T \times S} \quad (\text{T - Pair})$$

$$\frac{\Gamma \vdash t : T \times S}{\Gamma \vdash t.1 : T} \quad (\text{T - Proj1})$$

$$\frac{\Gamma \vdash t : T \times S}{\Gamma \vdash t.2 : S} \quad (\text{T - Proj2})$$

- Какой логической связке соответствует тип пары по Карри-Говарду?
- Синтаксис пар легко обобщается до кортежей и далее до записей. В последнем случае нужно лишь добавить синтаксис для меток полей и договориться о том, учитывать или не учитывать порядок.

Новые синтаксические формы и категории

```
p ::= x
    {p,p}
t ::= ...
    let p = t in t
```

Правила сопоставления

$$\text{match}(x, v) = [x \mapsto v] \quad (\text{M} - \text{Var})$$
$$\frac{\text{match}(p_1, v_1) = \sigma_1 \quad \text{match}(p_2, v_2) = \sigma_2}{\text{match}(\{p_1, p_2\}, \{v_1, v_2\}) = \sigma_1 \circ \sigma_2} \quad (\text{M} - \text{Pair})$$

Сопоставление с переменной всегда успешно, в противном случае возможна неудача: $\text{match}(\{x, y\}, 5)$.

A $\text{match}(x, \{5, \text{true}\})?$ $\text{match}(\{x, x\}, \{5, \text{true}\})?$

Новые правила вычисления

$$\text{let } p = v \text{ in } t \longrightarrow \text{match}(p, v)t \quad (\text{E - LetV})$$

$$\frac{t_1 \longrightarrow t'_1}{\text{let } p = t_1 \text{ in } t_2 \longrightarrow \text{let } p = t'_1 \text{ in } t_2} \quad (\text{E - Let})$$

Ясно, что сопоставление с образцом тоже легко расширяется на кортежи и записи.

Отношение типизации для образцов (\Rightarrow) порождает контекст.

Правила типизации для образцов

$$\vdash x : T \Rightarrow x : T \quad (\text{P - Var})$$

$$\frac{\vdash p_1 : T_1 \Rightarrow \Gamma_1 \quad \vdash p_2 : T_2 \Rightarrow \Gamma_2}{\vdash \{p_1, p_2\} : T_1 \times T_2 \Rightarrow \Gamma_1, \Gamma_2} \quad (\text{P - Pair})$$

Новое правило типизации

$$\frac{\Gamma \vdash t_1 : T_1 \quad \vdash p : T_1 \Rightarrow \Delta \quad \Gamma, \Delta \vdash t_2 : T_2}{\Gamma \vdash \text{let } p = t_1 \text{ in } t_2 : T_2} \quad (\text{T - Let})$$

Новые синтаксические формы

```
t ::= ...
    inl t
    inr t
    case t of inl x  $\Rightarrow$  t | inr x  $\Rightarrow$  t
v ::= ...
    inl v
    inr v
T ::= ...
    T + T
```

Новые правила вычисления

$$\text{case (inl } v) \text{ of inl } x_1 \Rightarrow t_1 \mid \text{inr } x_2 \Rightarrow t_2 \longrightarrow [x_1 \mapsto v]t_1 \quad (\text{E - CaseInl})$$

$$\text{case (inr } v) \text{ of inl } x_1 \Rightarrow t_1 \mid \text{inr } x_2 \Rightarrow t_2 \longrightarrow [x_2 \mapsto v]t_2 \quad (\text{E - CaseInr})$$

$$\frac{t \longrightarrow t'}{\text{case } t \text{ of inl } x_1 \Rightarrow t_1 \mid \text{inr } x_2 \Rightarrow t_2 \longrightarrow \text{case } t' \text{ of inl } x_1 \Rightarrow t_1 \mid \text{inr } x_2 \Rightarrow t_2} \quad (\text{E - Case})$$

$$\frac{t \longrightarrow t'}{\text{inl } t \longrightarrow \text{inl } t'} \quad (\text{E - Inl})$$

$$\frac{t \longrightarrow t'}{\text{inr } t \longrightarrow \text{inr } t'} \quad (\text{E - Inr})$$

Новые правила типизации

$$\frac{\Gamma \vdash t : T}{\Gamma \vdash \text{inl } t : T + S} \quad (\text{T - Inl})$$

$$\frac{\Gamma \vdash t : S}{\Gamma \vdash \text{inr } t : T + S} \quad (\text{T - Inr})$$

$$\frac{\Gamma \vdash t : T + S \quad \Gamma, x_1 : T \vdash t_1 : R \quad \Gamma, x_2 : S \vdash t_2 : R}{\Gamma \vdash \text{case } t \text{ of inl } x_1 \Rightarrow t_1 \mid \text{inr } x_2 \Rightarrow t_2 : R} \quad (\text{T - Case})$$

- Какой логической связке соответствует тип суммы по Карри-Говарду?
- Синтаксис суммы легко обобщается до типов-вариантов: метки полей заменяют теги `inl` и `inr`. Другое название — непересекающиеся объединения (disjoint union).

- Суммы нарушают единственность типизации, которая до сих пор имела место: `inl true : Bool + Nat` и `inl true : Bool + Bool`.
- Простейшее решение — обязать программиста писать конструкцию `as` в этом случае.
- Альтернативы — оставлять другое слагаемое в типе неопределенным или описывать все допустимые для него типы единообразно.

- 1 Соответствие Карри-Говарда
- 2 Простейшие расширения
- 3 Простые расширения
- 4 Рекурсия общего вида

- Рекурсия общего вида в бестиповом исчислении определяется с помощью комбинатора неподвижной точки.
- Наиболее известен комбинатор Карри
 $Y = \lambda f. (\lambda x. f(x x))(\lambda x. f(x x)).$
- Однако он не годится для семантики с вызовом по значению. (Почему?)
- Для нее подходит комбинатор Плоткина
 $Z = \lambda f. (\lambda x. f(\lambda y. x x y))(\lambda x. f(\lambda y. x x y)).$
- Проверьте, что это (1) комбинатор неподвижной точки, (2) годный для стратегии вызова по значению.

- Комбинатор неподвижной точки невозможно определить так, чтобы он имел допустимый тип в нашей системе. Введем его как примитив:

Новые синтаксические формы

```
t ::= ...  
    fix t
```

Новые правила вычисления

$$\text{fix } (\lambda x : T. t) \longrightarrow [x \mapsto \text{fix } (\lambda x : T. t)] t \quad (\text{E - FixBeta})$$
$$\frac{t \longrightarrow t'}{\text{fix } t \longrightarrow \text{fix } t'} \quad (\text{E - Fix})$$

Пример использования `fix`

```
ff = λie:Nat → Bool.
```

```
  λx:Nat.
```

```
    if iszero x then true
```

```
    else if iszero (pred x) then false
```

```
    else ie (pred (pred x));
```

```
▷ ff : (Nat → Bool) → Nat → Bool
```

```
iseven = fix ff;
```

```
▷ iseven : Nat → Bool
```

```
iseven 7;
```

```
▷ false : Bool
```

Новое правило типизации

$$\frac{\Gamma \vdash t : T \rightarrow T}{\Gamma \vdash \text{fix } t : T} \quad (T - \text{Fix})$$

- Тип $\text{fix} : (T \rightarrow T) \rightarrow T$ не является тавтологией пропозициональной логики; его добавление делает систему логически неконсистентной.
- Каждый тип становится населенным. Например, расходящимся термом `diverge unit`:

Семейство функций `diverge` (своя для каждого T)

```
diverge =  $\lambda\_:\text{Unit}.$  fix ( $\lambda x:T.x$ );  
▷ diverge : Unit  $\rightarrow$  T
```

- Для связывания переменной с результатом рекурсивного вызова удобен синтаксический сахар:

Новая производная форма

$$\text{letrec } x : T = t_1 \text{ in } t_2 \equiv \text{let } x = \text{fix } (\lambda x : T. t_1) \text{ in } t_2$$

Пример использования letrec

```
letrec iseven : Nat → Bool =  
  λx:Nat.  
    if iszero x then true  
    else if iszero (pred x) then false  
    else iseven (pred (pred x))  
in  
  iseven 7;  
▷ false : Bool
```