Функциональное программирование
Лекция 1. Лямбда-исчисление

Денис Николаевич Москвин

СПбАУ РАН

05.09.2017

План лекции

1. Функциональное vs императивное программирование
2. Введение в λ-исчисление
3. Чистое λ-исчисление: термы
4. Эквивалентность термов
5. Расширения чистого λ-исчисления

План лекции

Императивное программирование

Императивное программирование: вычисление описывается в терминах инструкций, изменяющих состояние вычислителя.
В императивных программах:

- Состояние изменяется инструкциями присваивания значений изменяемым переменным: `v = E`
- Инструкции исполняются последовательно `C1; C2; C3`
- Есть механизм условного исполнения: инструкции `if`, `switch`
- Есть механизм циклов: инструкции `while`, `for`

Иногда говорят про стиль фон Неймана (термин Джона Бэкуса)

Императивное программирование: пример

- Факториал числа `n` для фон-неймановского языка

```
res = 1;
for (i = n; i > 0; i--)
   res = res * i;
```

- Выполнение программы: переход вычислителя из начального состояния в конечное с помощью последовательных инструкций.
- Часть конечного состояния (`res`) интерпретируется как результат вычислений.

Функциональное программирование

Функциональная программа — *выражение*, её выполнение — вычисление (*редукция*) этого выражения.

- Нет состояний — нет *изменяемых* переменных
- Нет переменных — нет присваивания
- Нет циклов, поскольку нет различий между итерациями
- Последовательность не важна, поскольку выражения независимы

Вместо этого есть:

- Рекурсия — замена циклов
- Функции высших порядков (HOF)
- Сопоставление с образцом

Все функции — чистые, то есть зависят только от значений параметров.

Функциональное программирование: пример

Факториал числа `n` для функционального языка

```
factorial n = if n == 0 then 1 else n * factorial (n - 1)
```

Выполнение программы: редукция выражения с помощью *подстановки* определений функций в местах их «вызова» с заменой формальных параметров на фактические.

```
factorial 3
 -> if 3 == 0 then 1 else 3 * factorial (3-1)
 -> 3 * factorial (3-1)
 -> 3 * if (3-1) == 0 then 1 else (3-1) * factorial ((3-1)-1)
 -> ...
```

Реализация может быть эффективнее, но «подстановочная» семантика должна сохраняться.

План лекции

λ-исчисление

- λ-исчисление — формальная система, лежащая в основе функционального программирования.
- Разработано Алонзо Чёрчем в 1930-х для формализации и анализа понятия вычислимости.
- Имеет бестиповую и множество типизированных версий.
- Дает возможность компактно описывать семантику вычислительных процессов.

Неформальное введение в λ-исчисление

В λ-исчислении имеются два способа строить выражения:

- *применение (аппликация, application)*;
- *абстракция (abstraction)*.

Нотация применения F к X:

$$F\,X$$

- С точки зрения программиста: F (алгоритм) применяется к X (входные данные).
- Однако явного различия между алгоритмами и данными нет, например, допустимо самоприменение:

$$F\,F$$

Неформальное введение в λ-исчисление: абстракция

- Пусть $M \equiv M[x]$ — выражение, (возможно) содержащее x. Тогда абстракция

$$\lambda x.\, M$$

обозначает функцию

$$x \mapsto M[x],$$

то есть каждому x сопоставляется $M[x]$.

- Лямбда-абстракция — способ задать неименованную (анонимную) функцию.
- Если x в $M[x]$ отсутствует, то $\lambda x.\, M$ — константная функция со значением M.

Неформальное введение в λ-исчисление: $=_\beta$

- Применение и абстракция работают совместно:

$$\underbrace{(\lambda x.\, 2 \cdot x + 8)}_{F}\ \underbrace{17}_{X} \;=\; 2 \cdot 17 + 8 \quad (= 42).$$

- То есть $(\lambda x.\, 2 \cdot x + 8)\ 17$ — применение функции $x \mapsto 2 \cdot x + 8$ к аргументу 17, дающее в результате $2 \cdot 17 + 8$.
- В общем случае имеем β-эквивалентность

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

где $M[x := N]$ обозначает подстановку N вместо x в M.

- В *чистом* λ-исчислении нет ничего, кроме применения, абстракции и β-эквивалентности.

План лекции

Термы чистого λ-исчисления

Определение

Множество λ-***термов*** Λ индуктивно строится из переменных $V = \{x, y, z, \ldots\}$ с помощью применения и абстракции:

$$\begin{aligned} x \in V &\Rightarrow x \in \Lambda \\ M, N \in \Lambda &\Rightarrow (M\,N) \in \Lambda \\ M \in \Lambda, x \in V &\Rightarrow (\lambda x.\, M) \in \Lambda \end{aligned}$$

- В абстрактном синтаксисе

$$\Lambda \;::=\; V \mid (\Lambda\,\Lambda) \mid (\lambda V.\, \Lambda)$$

- **Соглашение.** Произвольные термы пишем заглавными буквами, переменные — строчными.

Термы (примеры)

Примеры λ-термов

$$x$$

$$(x\,z)$$

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

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

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

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

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

В этом списке каждый следующий терм содержит предыдущие в качестве *подтерма*.

Термы (соглашения)

Общепринятыы следующие соглашения:

- Внешние скобки опускаются.
- Применение ассоциативно *влево*:

$$\mathrm{F\,X\,Y\,Z} \quad \text{обозначает} \quad (((\mathrm{F\,X})\,\mathrm{Y})\,\mathrm{Z})$$

- Абстракция ассоциативна *вправо*:

$$\lambda x\,y\,z.\,\mathrm{M} \quad \text{обозначает} \quad (\lambda x.\,(\lambda y.\,(\lambda z.\,(\mathrm{M}))))$$

- Тело абстракции простирается вправо насколько это возможно:

$$\lambda x.\,\mathrm{M\,N\,K} \quad \text{обозначает} \quad \lambda x.\,(\mathrm{M\,N\,K})$$

Термы (примеры)

Те же примеры, что и выше, но с использованием соглашений

$$
\begin{aligned}
x &= x \\
(x\,z) &= x\,z \\
(\lambda x.\,(x\,z)) &= \lambda x.\,x\,z \\
((\lambda x.\,(x\,z))\,y) &= (\lambda x.\,x\,z)\,y \\
(\lambda y.\,((\lambda x.\,(x\,z))\,y)) &= \lambda y.\,(\lambda x\,.\,x\,z)\,y \\
((\lambda y.\,((\lambda x.\,(x\,z))\,y))\,w) &= (\lambda y.\,(\lambda x\,.\,x\,z)\,y)\,w \\
(\lambda z.\,(\lambda w.\,((\lambda y.\,((\lambda x.\,(x\,z))\,y))\,w))) &= \lambda z\,w.\,(\lambda y.\,(\lambda x\,.\,x\,z)\,y)\,w
\end{aligned}
$$

Свободные и связанные переменные

Абстракция $\lambda x\,.\,M[x]$ *связывает* дотоле *свободную* переменную x в терме M.

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

Переменные x и y — связанные, а z и w — свободные.

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

Переменная x — связанная (дважды!) и свободная, а z — свободная.

Свободные и связанные переменные (2)

Определение

Множество $\mathrm{FV}(T)$ ***свободных (free) переменных*** в λ-терме T:

$$\begin{aligned} \mathrm{FV}(x) &= \{x\}; \\ \mathrm{FV}(M\,N) &= \mathrm{FV}(M) \cup \mathrm{FV}(N); \\ \mathrm{FV}(\lambda x.\, M) &= \mathrm{FV}(M) \setminus \{x\}. \end{aligned}$$

Определение

Множество $\mathrm{BV}(T)$ ***связанных (bound) переменных*** в терме T:

$$\begin{aligned} \mathrm{BV}(x) &= \varnothing; \\ \mathrm{BV}(M\,N) &= \mathrm{BV}(M) \cup \mathrm{BV}(N); \\ \mathrm{BV}(\lambda x.\, M) &= \mathrm{BV}(M) \cup \{x\}. \end{aligned}$$

Комбинаторы

Определение

M — ***замкнутый*** λ***-терм*** (или ***комбинатор***), если $FV(M) = \varnothing$. Множество замкнутых λ-термов обозначается Λ^0.

Классические комбинаторы:

$$
\begin{aligned}
\mathbf{I} &= \lambda x.\, x \\
\boldsymbol{\omega} &= \lambda x.\, x\, x \\
\boldsymbol{\Omega} &= \boldsymbol{\omega}\, \boldsymbol{\omega} = (\lambda x.\, x\, x)(\lambda x.\, x\, x) \\
\mathbf{K} &= \lambda x\, y.\, x \\
\mathbf{K}_* &= \lambda x\, y.\, y \\
\mathbf{C} &= \lambda f\, x\, y.\, f\, y\, x \\
\mathbf{B} &= \lambda f\, g\, x.\, f\, (g\, x) \\
\mathbf{S} &= \lambda f\, g\, x.\, f\, x\, (g\, x)
\end{aligned}
$$

Переименование связанных переменных

Имена связанных переменных не важны: их можно (почти) безболезненно переименовывать:

$$\begin{aligned} \mathbf{I} &= \lambda x.\, x = \lambda y.\, y = \lambda z.\, z \\ \mathbf{B} &= \lambda f\, g\, x.\, f\,(g\, x) = \lambda u\, v\, z.\, u\,(v\, z) \end{aligned}$$

Это верно в том смысле, что термы с переименованиями при β-преобразованиях дают один и тот же результат:

$$\begin{aligned} (\lambda x.\, x)\, N &=_\beta N \\ (\lambda y.\, y)\, N &=_\beta N \\ (\lambda z.\, z)\, N &=_\beta N \end{aligned}$$

Термы, отличающиеся только именами связанных переменных, называют α-эквивалентными.

Функции нескольких переменных, каррирование

- Шейнфинкель (1924): функция $\mathfrak{n}$ аргументов может быть записана как $\mathfrak{n}$-элементная последовательность функций одного аргумента.
- Пусть $\varphi[x, y]$ — терм, содержащий свободные x и y. Введём, путём последавательных абстракций

$$\Phi_x = \lambda y.\, \varphi[x, y]$$
$$\Phi = \lambda x.\, \Phi_x = \lambda x.\, (\lambda y.\, \varphi[x, y]) = \lambda x\, y.\, \varphi[x, y]$$

- Тогда применение этого Φ к произвольным X и Y может быть выполнена последовательно

$$\Phi\, X\, Y = (\Phi\, X)\, Y = \Phi_X\, Y = (\lambda y.\, \varphi[X, y])\, Y = \varphi[X, Y].$$

- Переход от функции нескольких аргументов (функции над кортежем) к функции, принимающей аргументы «по одному» называется *каррированием*.

Комбинáторная логика

- Комбинаторы можно определить как примитивы, задав их поведение на аргументах-переменных:

Классические комбинаторы:

$$\begin{aligned} \mathbf{I}\,x &= x \\ \boldsymbol{\omega}\,x &= x\,x \\ \mathbf{K}\,x\,y &= x \\ \mathbf{S}\,f\,g\,x &= f\,x\,(g\,x) \end{aligned}$$

- Отношение эквивалентности задается подстановкой определения с заменой формальных аргументов на фактические:

$$\boldsymbol{\omega}\,\mathbf{I} = \mathbf{I}\,\mathbf{I} = \mathbf{I}.$$

- Выбрав базис (например $\mathbf{S}, \mathbf{K}$), получим исчисление, эквивалентное λ-исчислению.

План лекции

Подстановка терма

Определение

$M[x := N]$ обозначает ***подстановку*** N вместо свободных вхождений x в M. Правила подстановки:

$$\begin{aligned}
x[x := N] &= N; \\
y[x := N] &= y; \\
(P\,Q)[x := N] &= (P[x := N])\,(Q[x := N]); \\
(\lambda y.\,P)[x := N] &= \lambda y.\,(P[x := N]),\ y \notin FV(N); \\
(\lambda x.\,P)[x := N] &= (\lambda x.\,P).
\end{aligned}$$

Подразумевается, что $x \not\equiv y$.

Пример

$$((\lambda x.\,(\lambda x.\,x\,z)\,x)\,x)[x := N] = (\lambda x.\,(\lambda x.\,x\,z)\,x)\,N$$

Проблема захвата переменной

Неприятность: $(\lambda y.\, x\, y)[x := y]$ ($y \in \mathrm{FV}(N)$ в четвёртом правиле).

Соглашение Барендрегта

Имена связанных переменных всегда будем выбирать так, чтобы они отличались от имён свободных переменных.

Пример

Вместо

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

будем писать

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

Тогда можно использовать подстановку без оговорки о свободных и связанных переменных.

Лемма подстановки

Лемма подстановки

Пусть $M, N, L \in \Lambda$. Предположим $x \not\equiv y$ и $x \notin FV(L)$. Тогда

$$M[x := N][y := L] \equiv M[y := L][x := N[y := L]].$$

Доказательство

Нудная индукция по всем 5 случаям. ■

Преобразования (конверсии): β

Основная схема аксиом для λ-исчисления

Для любых $M, N \in \Lambda$

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

- Логические аксиомы и правила:

$$\begin{aligned}
&M \;=\; M;\\
&M = N \;\Rightarrow\; N = M;\\
&M = N, N = L \;\Rightarrow\; M = L;\\
&M = M' \;\Rightarrow\; MZ = M'Z;\\
&M = M' \;\Rightarrow\; ZM = ZM';\\
&M = M' \;\Rightarrow\; \lambda x.\, M = \lambda x.\, M' \quad (\text{правило } \xi).
\end{aligned}$$

- Если $M = N$ доказуемо в λ-исчислении, пишут $\lambda \vdash M = N$.

Преобразования (конверсии): α

Иногда вводят схему аксиом α-преобразования:

$$\lambda x.\, M = \lambda y.\, M[x := y] \quad (\alpha)$$

в предположении, что $y \notin \mathrm{FV}(M)$.
Для рассуждений достаточно соглашения Барендрегта, но для компьютерной реализации α-преобразование полезно:

Пример

Пусть $\omega = \lambda x.\, x\, x$ и $1 = \lambda y\, z.\, y\, z$. Тогда

$$\begin{aligned}
\omega\, 1 &= (\lambda x.\, x\, x)(\lambda y\, z.\, y\, z) \\
&=_\beta (\lambda y\, z.\, y\, z)(\lambda y\, z.\, y\, z) \\
&=_\beta \lambda z.\, (\lambda y\, z.\, y\, z)\, z \\
&=_\alpha \lambda z.\, (\lambda y\, z'.\, y\, z')\, z \\
&=_\beta \lambda z\, z'.\, z\, z' \\
&=_\alpha \lambda y\, z'.\, y\, z' \\
&=_\alpha \lambda y\, z.\, y\, z \;=\; 1
\end{aligned}$$

Индексы Де Брауна (De Bruijn)

- ***Индексы Де Брауна (De Bruijn)*** представляют альтернативный способ представления термов.
- Связанные переменные не именуются, а индексируются, индекс показывает, сколько лямбд «назад» переменная была связана:

$$\begin{aligned} \lambda x.\,(\lambda y.\,x\,y) &\leftrightarrow \lambda\,(\lambda\,2\,1) \\ \lambda x.\,x\,(\lambda y.\,x\,y\,y) &\leftrightarrow \lambda\,1\,(\lambda\,2\,1\,1) \end{aligned}$$

- Свободные переменные при этом получают индексы, превышающие число лямбд слева:

$$\lambda x.\,z\,x\,y \leftrightarrow \lambda\,3\,1\,2$$

- При таком представлении все α-эквивалентные термы кодируются одинаково, а коллизий захвата переменной не возникает.

Преобразования (конверсии): η

- Схема аксиом η-преобразования:

$$\lambda x.\, M\, x = M \qquad (\eta)$$

  в предположении, что $x \notin \mathrm{FV}(M)$.

- Смысл этой эквивалентности в том, что аппликативное поведение термов слева и справа от знака равенства одинаково; для произвольного N верно

$$(\lambda x.\, M\, x)\, N =_\beta M\, N$$

Преобразования (конверсии): η

- η-преобразование обеспечивает принцип ***экстенсиональности***: две функции считаются экстенсионально эквивалентными, если они дают одинаковый результат при любом одинаковом вводе:

$$\forall N \ : \ F\,N = G\,N.$$

- Выбирая $y \notin FV(F) \cup FV(G)$, получаем ($\xi$, затем η)

$$\begin{aligned} F\,y &= G\,y \\ \lambda y.\,F\,y &= \lambda y.\,G\,y \\ F &= G \end{aligned}$$

- В Хаскелле так называемый «бесточечный» стиль записи основан на η-преобразовании.

План лекции

Расширения чистого λ-исчисления

- Можно расширить множество λ-термов константами:

$$\Lambda(\mathbb{C}) \quad ::= \quad \mathbb{C} \mid V \mid \Lambda(\mathbb{C})\,\Lambda(\mathbb{C}) \mid \lambda V.\,\Lambda(\mathbb{C})$$

- Например, $\mathbb{C} = \{\mathbf{true}, \mathbf{false}\}$.
- Но нам ещё нужно уметь их использовать. Поэтому лучше

$$\mathbb{C} = \{\mathbf{true}, \mathbf{false}, \mathbf{not}, \mathbf{and}, \mathbf{or}\}$$

- И всё равно, помимо констант нужны дополнительные правила, описывающие работу с ними. Какие?

δ-преобразование: пример

- Всем известные:

$$\begin{aligned}
\mathbf{not\ true} &=_\delta \mathbf{false} \\
\mathbf{not\ false} &=_\delta \mathbf{true}
\end{aligned}$$

$$\begin{aligned}
\mathbf{and\ true\ true} &=_\delta \mathbf{true} \\
\mathbf{and\ true\ false} &=_\delta \mathbf{false} \\
\mathbf{and\ false\ true} &=_\delta \mathbf{false} \\
\mathbf{and\ false\ false} &=_\delta \mathbf{false} \\
&\dots
\end{aligned}$$

- «Внешние» функции над константами порождают новые правила преобразований.

δ-преобразование: обобщение

- Если на множестве термов X (обычно $X \subseteq \mathbb{C}$) задана «внешняя» функция $f : X^k \to \Lambda(\mathbb{C})$, то для неё добавляем δ-правило:
  - выбираем незанятую константу δ_f;
  - для $M_1, \ldots, M_k \in X$ добавляем правило сокращения

$$\delta_f \; M_1 \; \ldots \; M_k \; =_\delta \; f(M_1, \ldots, M_k)$$

- Для одной f — не одно правило, а целая *схема* правил.
- Например, для $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ схемы правил:

$$\begin{aligned}
\mathbf{plus}\ m\ n &=_\delta m + n \\
\mathbf{mult}\ m\ n &=_\delta m \times n \\
\mathbf{equal}\ n\ n &=_\delta \mathbf{true} \\
\mathbf{equal}\ m\ n &=_\delta \mathbf{false},\ \text{если } m \neq n
\end{aligned}$$