Курс: Функциональное программирование

Лекция 1. Лямбда-исчисление

Денис Николаевич Москвин

16.09.2011

Кафедра математических и информационных технологий
Санкт-Петербургского академического университета

План лекции

- Функциональное vs императивное программирование.
- Лямбда-исчисление.
- Подстановка и преобразования.
- Теорема о неподвижной точке.

План лекции

- Функциональное vs императивное программирование.
- Лямбда-исчисление.
- Подстановка и преобразования.
- Теорема о неподвижной точке.

Императивное программирование

Суть: меняем состояние с помощью инструкций

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

Иногда говорят про стиль фон Неймана (Джон Бэкус)

Императивное программирование: пример

Скалярное произведение для фон-нейманновского языка

```
res = 0;
for (i = 0; i < n; ++i)
   res = res + a[i] * b[i];
```

Сколько тут состояний и их изменений?

Абстрактное представление (пока забываем про I/O):

$$\sigma_0 \to \sigma_1 \to \sigma_1 \to \ldots \to \sigma_n$$

Выполнение программы: переход из начального *состояния* в конечное с помощью *инструкций*.

Функциональное программирование

Функциональная программа — выражение, её выполнение — вычисление этого выражения.

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

Вместо этого есть:

- Рекурсия — вместо циклов
- Функции высших порядков (HOF)

Функциональное программирование: пример

Скалярное произведение для функционального языка

```
innerProduct = (sum .) . zipWith (*)
```

Абстрактное представление:

$$f_n(f_{n-1}(\ldots f_2(f_1(\sigma_0))\ldots))$$

Выполнение программы: *вычисление выражения*.

```
innerProduct as bs -> ((sum .) . zipWith (*)) as bs
                   -> (sum .) (zipWith (*) as) bs
                   -> sum (zipWith (*) as bs)
                   -> ...
```

Функциональное программирование: преимущества и недостатки

Преимущества:

- Более ясная семантика
- Большая свобода при исполнении (например, поддержка параллельности)
- Большая выразительность
- Лучшая параметризация и модульность
- Удобство при работе с бесконечными данными

Недостатки:

- Ввод-вывод: сложно реализовать
- Интерактивные программы: сложно реализовать
- Проблемы с быстродействием

Чистота и побочные эффекты

```
bool flag;
int f (int n) {
    int retVal;
    if flag then retVal = 2 * n;
            else retVal = n;
    flag = ! flag;
    return retVal;
}
void test() {
    flag = true;
    printf("f(1) + f(2) = %d\n", f(1) + f(2));
    printf("f(2) + f(1) = %d\n", f(2) + f(1));
}
//   f(1) + f(2) = 4
//   f(2) + f(1) = 5
```

Чистота и побочные эффекты (2)

Функция `f` не является чистой (pure) математической функцией.

Её исполнение приводит к возникновению побочного эффекта (side effect).
(глобально доступное состояние + разрушающее присваивание)

В *чистом* функциональном программировании такие функции не используются.

План лекции

- Функциональное vs императивное программирование.
- Лямбда-исчисление.
- Подстановка и преобразования.
- Теорема о неподвижной точке.

Неформальное введение в λ-исчисление (1)

В λ-исчислении две операции: применение и абстракция.

Применение (Application):

$$\mathrm{F}\,\mathrm{X}$$

С точки зрения программиста:
F (алгоритм) применяется к X (входные данные).

(Допустимо самоприменение $\mathrm{F}\,\mathrm{F}$.)

Неформальное введение в λ-исчисление (2)

Абстракция (Abstraction):
Пусть $M \equiv M[x]$ — выражение, содержащее x. Тогда

$$\lambda x.\, M$$

обозначает функцию

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

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

Если x в $M[x]$ отсутствует, то $\lambda x.\, M$ — константная функция со значением M.

Неформальное введение в λ-исчисление (3)

Применение и абстракция работают совместно:

$$\underbrace{(\lambda x.\, 2 \times x + 1)}_{F}\ \underbrace{42}_{X} \;=\; 2 \times 42 + 1 \quad (= 85).$$

То есть $(\lambda x.\, 2 \times x + 1)\ 42$ — применение функции $x \mapsto 2 \times x + 1$ к аргументу 42, дающее в результате $2 \times 42 + 1$.

В общем случае имеем β-преобразование

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

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

Термы (1)

Множество λ-***термов*** Λ 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 (2)

Примеры λ-термов:

$$x$$

$$(x\,z)$$

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

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

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

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

Термы (3)

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*:

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

Термы (4)

Те же примеры, с использованием соглашений

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

Свободные и связанные переменные (1)

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

Примеры:

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

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)

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

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

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

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

Свободные и связанные переменные (3)

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

Классические комбинаторы:

$$\mathbf{I} \equiv \lambda x.\, x;$$

$$\omega \equiv \lambda x.\, x\, x; \qquad \mathbf{\Omega} \equiv \omega\, \omega = (\lambda x.\, x\, x)(\lambda x.\, x\, x);$$

$$\mathbf{K} \equiv \lambda x\, y.\, x; \qquad \mathbf{K}_* \equiv \lambda x\, y.\, y;$$

$$\mathbf{S} \equiv \lambda f\, g\, x.\, f\, x\, (g\, x);$$

$$\mathbf{B} \equiv \lambda f\, g\, x.\, f\, (g\, x).$$

Функции нескольких переменных, каррирование

Шонфинкель (1924): функции нескольких переменных могут быть описаны последовательным применением. Пусть $\varphi(x, y, z)$ — терм, зависящий от x, y, z.

$$\Phi_{x,y} = \lambda z.\, \varphi(x, y, z)$$
$$\Phi_x = \lambda y.\, \Phi_{x,y} = \lambda y.\, (\lambda z.\, \varphi(x, y, z))$$
$$\Phi = \lambda x.\, \Phi_x = \lambda x.\, (\lambda y.\, (\lambda z.\, \varphi(x, y, z))) = \lambda x\, y\, z.\, \varphi(x, y, z)$$

Тогда

$$\Phi\, X\, Y\, Z = ((\Phi\, X)\, Y)\, Z = (\Phi_x\, Y)\, Z = \Phi_{x,y}\, Z = \varphi(X, Y, Z).$$

Тождественное равенство термов

Имена *связанных* переменных не важны. Переименуем x в y:

$$\lambda x.\, M[x], \quad \lambda y.\, M[y]$$

Они ведут себя (при подстановках) одинаково:

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

Поэтому $P \equiv Q$ обозначает, что P и Q – это один и тот же терм с точностью до переименования связанных переменных. Например,

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

Иногда такое переименование называют α-преобразованием и пишут $P \equiv_\alpha Q$.

План лекции

- Функциональное vs императивное программирование.
- Лямбда-исчисление.
- Подстановка и преобразования.
- Теорема о неподвижной точке.

Подстановка (1)

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

Правила подстановки:

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

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

Пример:

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

Подстановка (2)

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 свободных переменных в терме (термах).

Например, 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

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

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:

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

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

Преобразования (конверсии): α и η

Иногда вводят:

- схему аксиом α-преобразования:

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

  в предположении, что $y \notin \mathrm{FV}(M)$;

- схему аксиом η-преобразования:

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

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

Преобразования (конверсии): α

Для рассуждений достаточно соглашения Барендрегта, но для компьютерной реализации α-преобразование полезно:

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

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

Преобразования (конверсии): α

Индексы Де Брауна (De Bruijn) представляют альтернативный способ представления термов.

Переменные не именуются, а нумеруются (индексируются), индекс показывает, сколько лямбд назад переменная была связана:

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

Преобразования (конверсии): η

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

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

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

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

Расширения чистого λ-исчисления

Можно расширить множество λ-термов константами:

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

Например, $\mathbb{C} = \{\textbf{true}, \textbf{false}\}$?

Но нам ещё нужно уметь их использовать. Поэтому лучше

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

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

δ-преобразование: пример

Всем известные:

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

$$\begin{aligned}
\textbf{and true true} &=_\delta \textbf{true} \\
\textbf{and true false} &=_\delta \textbf{false} \\
\textbf{and false true} &=_\delta \textbf{false} \\
\textbf{and false false} &=_\delta \textbf{false} \\
&\ldots
\end{aligned}$$

«Внешние» функции над константами порождают новые правила редукции.

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 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:

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

План лекции

- Функциональное vs императивное программирование.
- Лямбда-исчисление.
- Подстановка и преобразования.
- Теорема о неподвижной точке.

Термовые уравнения

Схема β-преобразования даёт возможность решать простейшие уравнения на термы.

Пример: найти F, такой что $\forall M, N, L \quad \lambda \vdash F\,M\,N\,L = M\,L\,(N\,L)$.

$$F\,M\,N\,L = M\,L\,(N\,L)$$
$$F\,M\,N = \lambda z.\, M\,z\,(N\,z)$$
$$F\,M = \lambda y.\, \lambda z.\, M\,z\,(y\,z)$$
$$F = \lambda x\,y\,z.\, x\,z\,(y\,z)$$

А если уравнение рекурсивное, например, $F\,M = M\,F$?

Теорема неподвижной точки (1)

Теорема. Для любого λ-терма F существует неподвижная точка:

$$\forall F \in \Lambda \; \exists X \in \Lambda \qquad \lambda \vdash FX = X$$

Док-во. Введем $W \equiv \lambda x.\, F\,(x\,x)$ и $X \equiv WW$. Тогда

$$X \equiv WW \equiv (\lambda x.\, F(x\,x))\,W = F\,(WW) \equiv FX \quad \blacksquare$$

Теорема. Существует комбинатор неподвижной точки

$$\mathbf{Y} \equiv \lambda f.\, (\lambda x.\, f\,(x\,x))(\lambda x.\, f\,(x\,x)),$$

такой что $\forall F \quad F(\mathbf{Y}\,F) = \mathbf{Y}\,F$.

Док-во. $\mathbf{Y}\,F \equiv (\lambda x.\, F\,(x\,x))(\lambda x.\, F\,(x\,x)) = F\,(\underbrace{(\lambda x.\, F\,(x\,x))(\lambda x.\, F\,(x\,x))}_{\mathbf{Y}\,F}) \equiv$

$F(\mathbf{Y}\,F) \quad \blacksquare$

Теорема неподвижной точки (2)

Y-комбинатор позволяет ввести рекурсию в λ-исчисление.

Факториал рекурсивно:

$$\mathsf{fac} = \lambda \mathrm{n}.\, \mathsf{iif}\,(\mathsf{iszro}\ \mathrm{n})\ \mathsf{1}\ (\mathsf{mult}\ \mathrm{n}\,(\mathsf{fac}\,(\mathsf{pred}\ \mathrm{n})))$$

Переписываем в виде

$$\mathsf{fac} = (\lambda \mathrm{f}\,\mathrm{n}.\, \mathsf{iif}\,(\mathsf{iszro}\ \mathrm{n})\ \mathsf{1}\ (\mathsf{mult}\ \mathrm{n}\,(\mathrm{f}\,(\mathsf{pred}\ \mathrm{n}))))\,\mathsf{fac}$$

Отсюда видно, что fac — неподвижная точка для функции $\mathsf{fac}' \equiv \lambda \mathrm{f}\,\mathrm{n}.\, \mathsf{iif}\,(\mathsf{iszro}\ \mathrm{n})\ \mathsf{1}\ (\mathsf{mult}\ \mathrm{n}\,(\mathrm{f}\,(\mathsf{pred}\ \mathrm{n})))$:

$$\mathsf{fac} = \mathbf{Y}\ \mathsf{fac}'$$

Теорема неподвижной точки (3)

Как работает $\mathsf{fac} \equiv \mathbf{Y}\,\mathsf{fac}'$?

$$
\begin{aligned}
\mathsf{fac}\ 3 &= (\mathbf{Y}\,\mathsf{fac}')\,3 \\
&= \mathsf{fac}'\,(\mathbf{Y}\,\mathsf{fac}')\,3 \\
&= \mathsf{IIF}\,(\mathsf{iszro}\ 3)\ 1\ (\mathsf{mult}\ 3\,((\mathbf{Y}\,\mathsf{fac}')\,(\mathsf{pred}\ 3))) \\
&= \mathsf{mult}\ 3\,((\mathbf{Y}\,\mathsf{fac}')\,2) \\
&= \mathsf{mult}\ 3\,(\mathsf{F}\,(\mathbf{Y}\,\mathsf{fac}')\,2) \\
&= \mathsf{mult}\ 3\,(\mathsf{mult}\ 2\,((\mathbf{Y}\,\mathsf{fac}')\,1)) \\
&= \mathsf{mult}\ 3\,(\mathsf{mult}\ 2\,(\mathsf{mult}\ 1\,((\mathbf{Y}\,\mathsf{fac}')\,0))) \\
&= \mathsf{mult}\ 3\,(\mathsf{mult}\ 2\,(\mathsf{mult}\ 1\ 1)) \\
&= 6
\end{aligned}
$$