Функциональное программирование
Lambda-исчисление

курс: «Парадигмы программирования"

доц. Нестеренко В.А.

λ-исчисление – абстрактная модель вычислений в функциональной парадигме программирования

- Сформулировано Алонзо Чёрчем (Alonzo Church) в 1930 году как теория исчисления анонимных (безымянных) функций.
- λ-исчисление предоставляет собой формальный метод представления функций и правила вывода значений функций.
- λ-исчисление является теоретическим базисом для функционального программирования.

Определение λ-выражений

1. Идентификатор: константа или переменная (x, alpha, t17, true, …) – λ-выражение.
2. Если x – переменная и M – λ-выражение, то конструкция (λx.M) тоже λ-выражение (абстракция).
3. Если M и N – λ-выражения, то конструкция (M N) тоже λ-выражение (применение).

Терминология: λ-выражение ≡ терм

Примеры λ-выражений

1. $(\lambda x.x) \rightarrow \lambda x.x$
2. $(\lambda x.(\lambda y.x)) \rightarrow \lambda xy.x$
3. $(\lambda x.(\lambda y.((+\ x)\ y))) \rightarrow \lambda xy.(+\ x\ y)$
4. $(\lambda f.(\ \lambda g.(\ \lambda x.(f\ (g\ x))))) \rightarrow \lambda fgx.(f\ (g\ x))$
5. $((\lambda x.(x\ x))\ (\lambda y.(y\ y))) \rightarrow \lambda x.(x\ x)\ \lambda y.(y\ y)$

Для упрощения записи λ-выражений скобки можно опускать если это не приводит к неоднозначности.

Следует учитывать:
применения ассоциативны слева: M N P соответствует ((M N) P)
абстракции ассоциативны справа: λx.λy.M обозначает (λx.(λy.M))

Связанные и свободные переменные

абстракция $\lambda x.M$ – определение функции

применение $(M\ N)$ – вызов функции

Переменная x в абстракции $\lambda x.M[x]$ – связанная переменная.

Переменная y в абстракции $\lambda x.M[y \neq x]$ – свободная переменная.

Связанные переменные – аналог формальных параметров функции.

Редукция λ-выражений

Вычисление λ-выражений – упрощение – редукция

1. Константа представляет сама себя и её невозможно преобразовать в более простое выражение.
2. α-преобразование: $\lambda x.M(x) \to \lambda y.M(y)$ x, y – связанные переменные
3. β-редукция: применение абстракций и замена связанных переменных фактическими значениями.

   Например: $(\lambda x.(* \ x \ x) \ 2) \to (* \ 2 \ 2)$, $(\lambda x.x \ M) \to M$.
4. δ-редукция: функция, имя которой представлено константой, преобразуется (редуцируется), при наличии достаточного числа аргументов, путём использования встроенных правил.

   Например: $(* \ 2 \ 2) \to 4$, $(\text{if } FALSE \ M \ N) \to N$.

Правила редукции следует применять столько раз, сколько это возможно:

$\lambda x.\lambda y.(+ \ x \ y) \ 7 \ 8 \to \lambda y.(+ \ 7 \ y) \ 8 \to (+ \ 7 \ 8) \to 15$

Редукция, продолжение

Процесс вычисления λ-выражения завершается, если к нему нельзя применить никакое правило редукции. В этом случае говорят, что выражение приведено к *нормальной форме*.

λx.x A → A

λx.λy.(* (- x y) (+ x y)) 8 7
→ λy.(* (- 8 y) (+ 8 y)) 7
→ (* (- 8 7) (+ 8 7))
→ 15

(λx.(x x)) (λy.(y y))
→ (λy.(y y)) (λy.(y y)) = (λx.(x x)) (λy.(y y))

Последний пример показывает, что редукция не всегда приводит выражение к нормальной форме.

Порядок редукции

Редекс – простейшее выражение для которого возможна редукция.
(λx.M N) – β-редекс.

(+ a b) – δ-редекс

Выражение *(λx.λy.y) ((λz.(z z)) (λz.(z z)))* содержит 2 редекса.

Применим β-редукцию к первому:
(λx.λy.y) ((λz.(z z)) (λz.(z z)))

→ *λy.y*

Применим β-редукцию ко второму:
(λx.λy.y) ((λz.(z z)) (λz.(z z)))

$=^{\alpha}$ *(λx.λy.y) ((λz.(z z)) (λx.(x x)))*

→ *(λx.λy.y) ((λx.(x x)) (λy.(y y)))*

$=^{\alpha}$ *(λx.λy.y) ((λz.(z z)) (λz.(z z)))*

Получили разные результаты: важен порядок применения редукции.

Порядок редукции, продолжение

Две стратегии:

Нормальный порядок редукции (НПР) предписывает первым преобразовывать самый левый из самых внешних редексов.

Аппликативный порядок редукции (АПР) предписывает первым преобразовывать самый левый из самых внутренних редексов.

Теорема. Если у λ-выражения есть нормальная форма, то применение нормального порядка редукции (НПР) на каждом этапе вычисления гарантирует достижение этой нормальной формы.

Самый внешний редекс – это редекс, который не содержится внутри никакого другого редекса.

Самый левый редекс – это редекс символ которого (или идентификатор встроенной функции для δ-редекса) расположен левее символов других редексов.

Порядок редукции

Для выражения

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

НПР:

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

$\rightarrow \lambda y.y$

АПР:

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

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

Сравнение НПР и АПР для приведённого примера показывает, что НПР соответствует схеме «ленивых» вычислений.

Порядок редукции

Ещё пример:

$\lambda z.(* \; z \; z) \; (+ \; 1 \; 2)$

НПР:

$\lambda z.(* \; z \; z) \; (+ \; 1 \; 2)$
$\rightarrow (* \; (+ \; 1 \; 2) \; (+ \; 1 \; 2))$
$\rightarrow (* \; 3 \; 3)$
$\rightarrow 9$

АПР:

$\lambda z.(* \; z \; z) \; (+ \; 1 \; 2)$
$\rightarrow \lambda z.(* \; z \; z) \; 3$
$\rightarrow 9$

АПР даёт более эффективную схему вычислений.

Комбинатор неподвижной точки

Комбинатор – λ-выражение не содержащее свободных переменных.

Неподвижная точка X для выражения *F* удовлетворяет условию: *F X = X* .

Y-комбинатор: $Y = \lambda h.(\lambda x.(h\ (x\ x))\ \lambda x.(h\ (x\ x)))$ – введён Х. Карри

$(Y\ F) = \lambda h.(\lambda x.(h\ (x\ x))\ \lambda y.(h\ (y\ y)))\ F$

$\rightarrow (\lambda x.(F\ (x\ x))\ \lambda y.(F\ (y\ y)))$

$\rightarrow (F\ (\lambda y.(F\ (y\ y))\ \lambda z.(F\ (z\ z)))$

$= (F\ (Y\ F))$ *(Y F) неподвижная точка для F*

T-комбинатор: $T = (\lambda x.\lambda y.(y\ (x\ x\ y)))\ (\lambda x.\lambda y.(y\ (x\ x\ y)))$ – введён А. Тьюрингом

$(T\ F) = (\lambda x.\lambda y.(y\ (x\ x\ y)))\ (\lambda x.\lambda y.(y\ (x\ x\ y)))\ F$

$\rightarrow (F\ ((\lambda x.\lambda y.(y\ (x\ x\ y)))\ (\lambda x.\lambda y.(y\ (x\ x\ y)))\ F))$

$= F\ (T\ F)$ *(T F) неподвижная точка для F*

Неподвижную точку *X* выражения *F* можно найти применив комбинатор неподвижной точки к выражению *F*: *X=(Y F)* или *X=(T F)* .

Тогда: *F (Y F) = (Y F)* или *F (T F) = (T F)*

Рекурсия в λ-исчислении

Начнём с выражения (некорректного в λ-исчислении) для факториала:

fact = λn.(if (zero n) 1 (n fact (- n 1)))*
→ *λf.λn.(if (zero n) 1 (* n f (- n 1))) fact = (F fact)*
где *F = λf.λn.(if (zero n) 1 (* n f (- n 1)))*

Мы получили *(F fact) = fact .* Отсюда следует, что *fact* – неподвижная точка выражения *F*: *fact = (Y F)*

(F = … и fact = … это не присваивание значений переменным, а условные обозначения выражений именами F и fact)

Вывод: если *F* некоторая λ-абстракции, то выражение *(Y F)* представляет рекурсивную схему вычислений с правилами задаваемыми F.

Y – комбинатор неподвижной точки.

Факториал, продолжение

Пусть: $F = \lambda f.\lambda n.(if\ (eq\ n\ 0)\ 1\ (*\ n\ (f\ (-\ n\ 1))))$ и $fact = (Y\ F)$.

$(Y\ F) = F\ (Y\ F)$, Y – комбинатор неподвижной точки.

$(fact\ N) = ((Y\ F)\ N) = ((F\ (Y\ F))\ N) = (F\ fact\ N)$

Конкретно:

$fact\ 4 = (Y\ F)\ 4 = F\ (Y\ F)\ 4 = \lambda f.\lambda n.(if\ (eq\ n\ 0)\ 1\ (*\ n\ (f\ (-\ n\ 1))))\ (Y\ F)\ 4$

$\rightarrow (*\ 4\ ((Y\ F)\ (-\ 4\ 1)))$

$\rightarrow *\ 4\ (F\ (Y\ F)\ 3)$

. . .

$\rightarrow *\ 4\ (*\ 3\ (F\ (Y\ F)\ 2))$

. . .

$\rightarrow *\ 4\ (*\ 3\ (*\ 2\ (F\ (Y\ F)\ 1)))$

. . .

$\rightarrow *\ 4\ (*\ 3\ (*\ 2\ (*\ 1\ (F\ (Y\ F)\ 0))))$

$\rightarrow *\ 4\ (*\ 3\ (*\ 2\ (*\ 1\ (\lambda f.\lambda n.(if\ (eq\ n\ 0)\ 1\ (*\ n\ (f\ (-\ n\ 1))))\ (Y\ F)\ 0))))$

$\rightarrow *\ 4\ (*\ 3\ (*\ 2\ (*\ 1\ 1)))$

$\rightarrow 24$

Куайн (quine) функция и λ-исчисление

quine-программа – программа, которая выдаёт на выходе точную копию своего исходного текста.

quine-функция – функция, которая в качестве значения возвращает точную копию своего определения.

Претендент:
(λx.(x x) λy.(y y)) → (λy.(y y) λz.(z z)) → … (редукция не завершается)

Lisp:
((lambda (x) (x x)) (lambda (y) (y y)))
(… Вывод с использованием трассировщика …)
((lambda (x) (list x (list (quote quote) x))) (quote (lambda (x) (list x (list (quote quote) x)))))

Куайн (quine) функция, продолжение

λ-исчисление:

Пусть Q – quine.

- Если Q в нормальной форме: $Q \to Q$, то это готовый quine (тривиально).
- Если Q не имеет нормальной формальный, то конечное значение выражения не определено.

Переопределим понятие:

будем считать Q quine-выражением если для любого M редукция (λx.Q M) приводит к результату λx.Q или (λx.Q M) → λx.Q.

Ответ: $Q = (\lambda x.\lambda y.(x\ x))\ (\lambda x.\lambda y.(x\ x))$

Проверка:

$$\lambda x.Q\ M = \lambda x.((\lambda z.\lambda y.(z\ z))\ (\lambda z.\lambda y.(z\ z)))\ M$$

$$\to (\lambda z.\lambda y.(z\ z))\ (\lambda z.\lambda y.(z\ z)))$$

$$\to \lambda y.((\lambda z.\lambda y.(z\ z))\ (\lambda z.\lambda y.(z\ z)))$$

$$= \lambda x.((\lambda z.\lambda y.(z\ z))\ (\lambda z.\lambda y.(z\ z))) = \lambda x.Q$$

Куайн (quine) функция, продолжение

λ-исчисление:

Задача: найдём такое Q, чтобы $(\lambda x.Q\ M) \rightarrow \lambda x.Q$

- ✓ Если Q не содержит переменной z, то $(\lambda z.Q\ M) \rightarrow Q$. Теперь нужно найти такое выражение Q, что $Q \rightarrow \lambda z.Q$
- ✓ Введём $f = \lambda t.(\lambda z.t)$, тогда $(f\ Q) = (\lambda t.(\lambda z.t)\ Q) \rightarrow \lambda z.Q$
- ✓ Если потребовать $(f\ Q)=Q$, то из $(f\ Q) \rightarrow \lambda z.Q$ следует $Q\rightarrow\lambda z.Q$. То есть Q должно быть неподвижной точкой f : $Q = (Y\ f)$

Явный вид:

$Q = (Y\ \ \lambda t.(\lambda z.t))$

$\rightarrow (\ (\lambda h.(\lambda x.(h\ (x\ x))\ \lambda x.(h\ (x\ x))))\ \ \lambda t.(\lambda z.t)\ )$

$\rightarrow (\ \lambda x.(\lambda t.(\lambda z.t)\ (x\ x))\ \lambda x.(\lambda t.(\lambda z.t)\ (x\ x))\ )$

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

Одно из представлений: $Q = (\lambda x.\lambda z.(x\ x)\ \lambda x.\lambda z.(x\ x))$