



Функциональное и логическое программирование

Лекция 6. Введение в функциональное программирование



Учебные вопросы

- Принципы и достоинства функционального программирования
- Принципы лямбда нотации. Понятие лямбда терма. Свободные и связанные переменные. Подстановка и преобразования
- Эквивалентность лямбда выражений. Экстенциональность. Редукция
- Теорема Чёрча-Россера. Комбинаторы



Принципы и достоинства функционального программирования

Определение парадигмы

Определение

Парадигма программирования (англ. *programming paradigm*) — совокупность идей и понятий, которые определяют общий стиль написания компьютерных программ, построения их структуры и отдельных элементов программной системы.

Цель парадигмы программирования:

- ▶ разделение программы на базовые составные элементы (напр., функции или объекты);
- ▶ определение модели преобразования данных;
- ▶ внедрение ограничений на используемые конструкции.

Принципы и достоинства функционального программирования

Классификация парадигм программирования





Принципы и достоинства функционального программирования

Императивное программирование

Определение

Императивное программирование (англ. *imperative programming*) — парадигма, согласно которой программа представляет собой последовательность действий, изменяющих *состояние программы*.

Определение

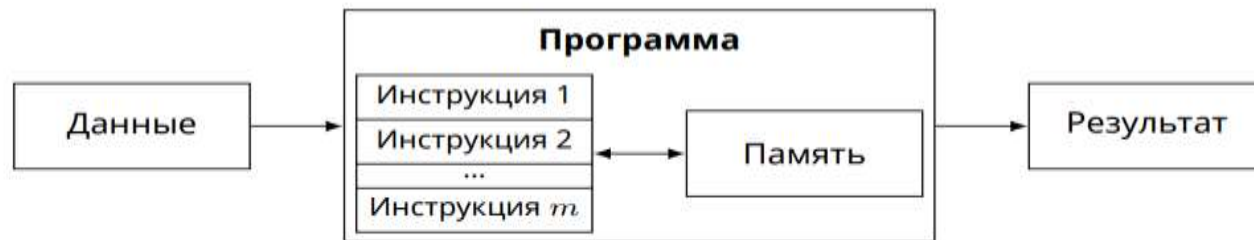
Состояние программы (англ. *program state*) — совокупность данных, связанных со всеми используемыми программой переменными в конкретный момент времени.

Область использования:

- ▶ системные программы;
- ▶ прикладные программы.

Принципы и достоинства функционального программирования

Выполнение императивной программы



Императивная программа использует именованные области памяти (переменные) для хранения состояния вычислений



Принципы и достоинства функционального программирования

Декларативное программирование

Определение

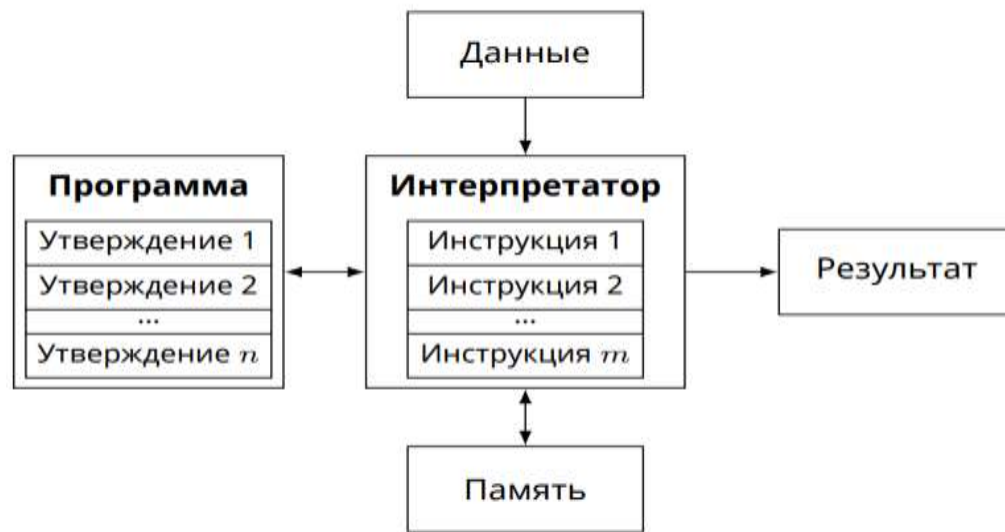
Декларативное программирование (англ. *declarative programming*) — парадигма, согласно которой программа представляет логику вычислений без описания прямой последовательности действий (действия определяются компилятором или интерпретатором).

Области использования:

- ▶ математическое моделирование;
- ▶ искусственный интеллект;
- ▶ анализ данных;
- ▶ наука.

Принципы и достоинства функционального программирования

Выполнение декларативной программы



Декларативная программа не взаимодействует напрямую с памятью, поручая эту работу интерпретатору



Принципы и достоинства функционального программирования

Функциональное программирование

Определение

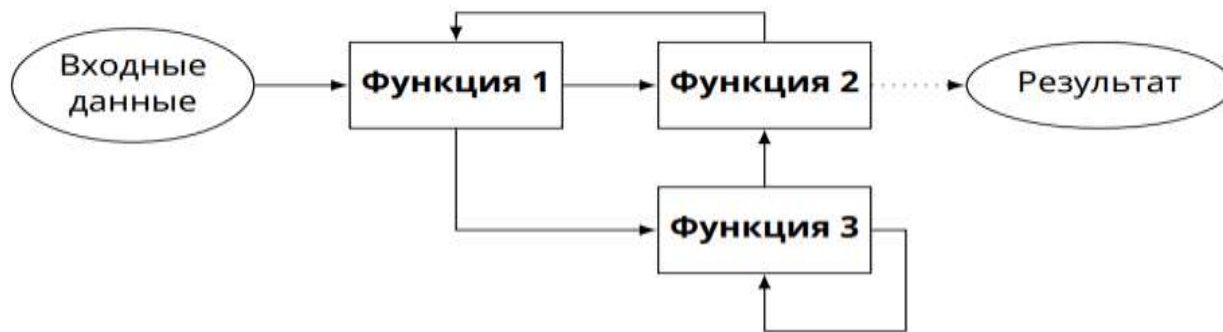
Функциональное программирование (англ. *functional programming*) — парадигма, согласно которой процесс исполнения программы представляется последовательностью вычислений значений для математических функций.

Особенности:

- ▶ отказ от явного хранения переменных (функции без побочных эффектов);
- ▶ ⇒ встроенная поддержка параллелизации, оптимизации и кэширования без необходимости действий со стороны программиста.

Принципы и достоинства функционального программирования

Функциональное программирование (продолжение)



Функции возвращают результат, иначе не меняя состояние программы (напр., через переменные).

Ключевые слова: чистая функция, прозрачность ссылок.



Принципы и достоинства функционального программирования

Функциональное программирование (продолжение)

Концепции:

- ▶ функции высших порядков — функции, которые возвращают другие функции или принимают функции в качестве аргументов;
- ▶ замыкание (англ. *closure*) — сохранение контекста функции при ее создании;
- ▶ рекурсия для создания циклов;
- ▶ ленивые вычисления (англ. *lazy evaluation*) — вычисление аргументов функций по мере необходимости (не при задании).

Языки программирования: Lisp, Scheme, Clojure, Erlang, Haskell, F#.

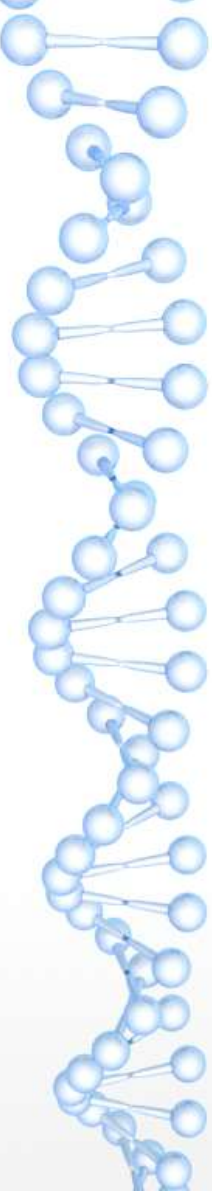


Принципы и достоинства функционального программирования

```
let rec fibonacci n =  
  match n with  
  | 1 | 2 -> 1  
  | _ -> fibonacci (n-1) + fibonacci (n-2)
```

```
let rec printFib n =  
  match n with  
  | 1 -> printf "%d, " (fibonacci (n))  
  | _ -> printFib (n-1)  
        printf "%d, " (fibonacci (n))
```

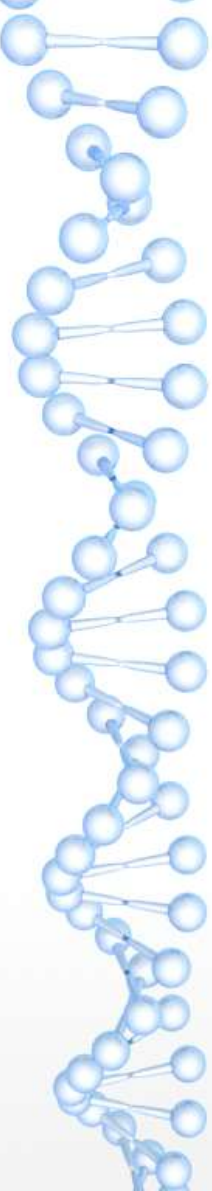
```
printFib(16)  
printfn "..."
```



Принципы лямбда нотации. Понятие лямбда терма

Основой λ -исчисления служит формальное понятие λ -термов, которые строятся из переменных и некоторого фиксированного множества констант при помощи операций применения (аппликации) функций и λ -абстракции. Это значит, что все возможные λ -термы разбиваются на четыре класса:

1. **Переменные:** обозначаются произвольными алфавитно-цифровыми строками; как правило, мы будем использовать в качестве имён буквы, расположенные ближе к концу латинского алфавита, например, x , y и z .
2. **Константы:** количество констант определяется конкретной λ -нотацией, иногда их нет вовсе. Мы будем также обозначать их алфавитно-цифровыми строками, как и переменные, отличая друг от друга по контексту.
3. **Комбинации:** применение функции s к аргументу t , где s и t представляют собой произвольные термы. Будем обозначать комбинации как $s\ t$, а их составные части называть «ратор» и «ранд» соответственно.²
4. **Абстракция** произвольного λ -терма s по переменной x (которая может как входить свободно в s , так и нет) имеет вид $\lambda x. s$.



Свободные и связанные переменные.

Подстановка и преобразования

В этом разделе мы формализуем интуитивное понятие свободных и связанных переменных, которое, между прочим, служит хорошим примером определения примитивно-рекурсивных функций. Интуитивно, вхождение переменной в заданный терм считается свободным, если оно не лежит в области действия соответствующей абстракции. Обозначим множество свободных переменных в терме s через $FV(s)$ и дадим его рекурсивное определение:

$$FV(x) = \{x\}$$

$$FV(c) = \emptyset$$

$$FV(st) = FV(s) \cup FV(t)$$

$$FV(\lambda x. s) = FV(s) - \{x\}$$

Аналогично вводится и понятие множества связанных переменных $BV(s)$:

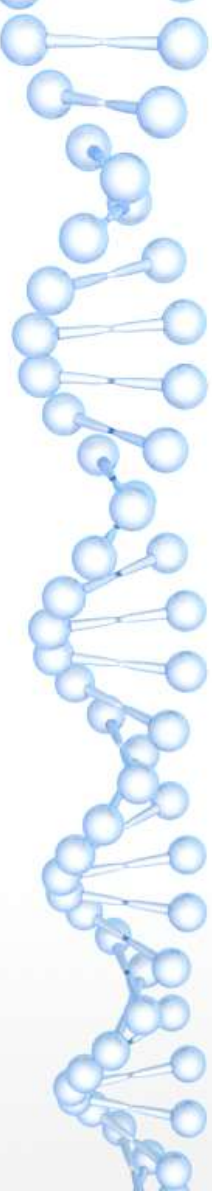
$$BV(x) = \emptyset$$

$$BV(c) = \emptyset$$

$$BV(st) = BV(s) \cup BV(t)$$

$$BV(\lambda x. s) = BV(s) \cup \{x\}$$

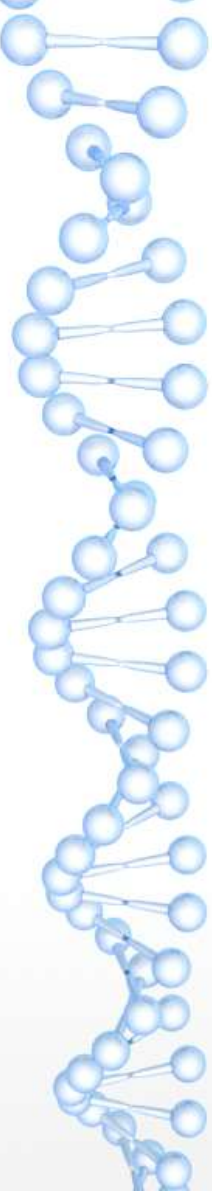
Например, если $s = (\lambda x y. x) (\lambda x. z x)$, то $FV(s) = \{z\}$ и $BV(s) = \{x, y\}$. Отметим, что в общем случае переменная может быть одновременно и свободной, и связанной в одном и том же терме, как это было показано выше. Воспользуемся структурной индукцией, чтобы продемонстрировать доказательство утверждений о свойствах λ -термов на примере следующей теоремы (аналогичные рассуждения применимы и ко множеству BV).



Свободные и связанные переменные. Подстановка и преобразования

Теорема 2.1 Для произвольного λ -терма s множество $FV(s)$ конечно.

Доказательство: Применим структурную индукцию. Очевидно, что для терма s , имеющего вид переменной либо константы, множество $FV(s)$ конечно по определению (содержит единственный элемент либо пусто соответственно). Если терм s представляет собой комбинацию t и u , то согласно индуктивному предположению, как $FV(t)$, так и $FV(u)$ конечны, в силу чего $FV(s) = FV(t) \cup FV(u)$ также конечно, как объединение двух конечных множеств. Наконец, если s имеет форму $\lambda x. t$, то по определению $FV(s) = FV(t) - \{x\}$, а $FV(t)$ конечно по индуктивному предположению, откуда следует, что $FV(s)$ также конечно, поскольку его мощность не может превышать мощности $FV(t)$. \square

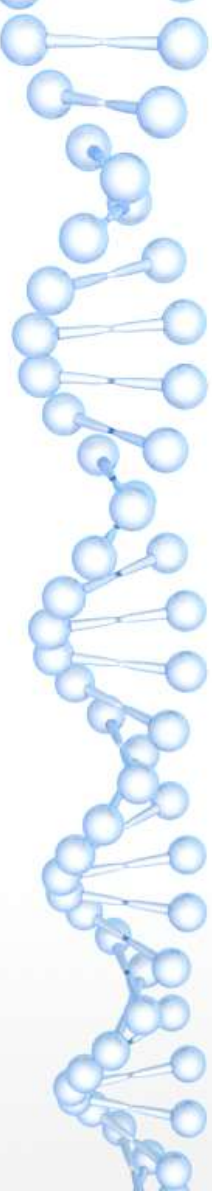


Свободные и связанные переменные. Подстановка и преобразования

Одним из правил, которые мы хотим формализовать, является соглашение о том, что λ -абстракция и применение функции представляют собой взаимно обратные операции. То есть, если мы возьмём терм $\lambda x. s$ и применим его как функцию к терму-аргументу t , результатом будет терм s , в котором все свободные вхождения переменной x заменены термом t . Для большей наглядности это действие принято обозначать $\lambda x. s[x]$ и $s[t]$ соответственно.

Однако, простое на первый взгляд понятие подстановки одного терма вместо переменной в другой терм на самом деле оказалось весьма коварным, так что даже некоторые выдающиеся логики не избежали ложных утверждений относительно его свойств. Подобный грустный опыт разочаровывает довольно сильно, ведь как мы говорили ранее, одним из привлекательных свойств формальных правил служит возможность их чисто механического применения.

Обозначим операцию подстановки терма s вместо переменной x в другой терм t как $t[s/x]$. Иногда можно встретить другие обозначения, например, $t[x:=s]$, $[s/x]t$, или даже $t[x/s]$. Мы полагаем, что предложенный нами вариант легче всего запомнить по аналогии с умножением дробей: $x[t/x] = t$. На первый взгляд, рекуррентное определение понятия подстановки выглядит так:



Свободные и связанные переменные. Подстановка и преобразования

$$x[t/x] = t$$

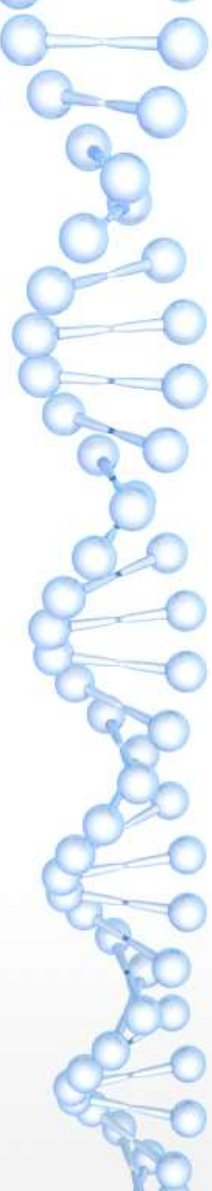
$$y[t/x] = y, \text{ если } x \neq y$$

$$c[t/x] = c$$

$$(s_1 \ s_2)[t/x] = s_1[t/x] \ s_2[t/x]$$

$$(\lambda x. s)[t/x] = \lambda x. s$$

$$(\lambda y. s)[t/x] = \lambda y. (s[t/x]), \text{ если } x \neq y$$



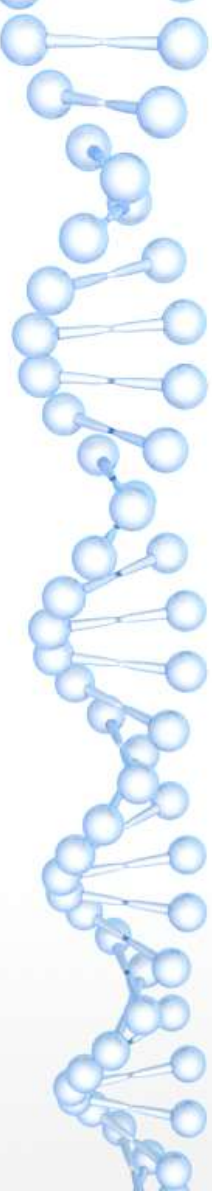
Свободные и связанные переменные. Подстановка и преобразования

К сожалению, это определение не совсем верно. Например, подстановка $(\lambda y. x + y)[y/x] = \lambda y. y + y$ не соответствует интуитивным ожиданиям от её результата.³ Исходный λ -терм интерпретировался как функция, прибавляющая x к своему аргументу, так что после подстановки мы ожидали получить функцию, которая прибавляет y , а на деле получили функцию, которая свой аргумент удваивает. Источником проблемы служит *захват* переменной y , которую мы подставляем, операцией $\lambda y. \dots$, которая связывает одноимённую переменную. Чтобы этого не произошло, связанную переменную требуется предварительно переименовать:

$$(\lambda y. x + y) = (\lambda w. x + w),$$

а лишь затем производить подстановку:

$$(\lambda w. x + w)[y/x] = \lambda w. y + w$$



Свободные и связанные переменные. Подстановка и преобразования

$$x[t/x] = t$$

$$y[t/x] = y, \text{ если } x \neq y$$

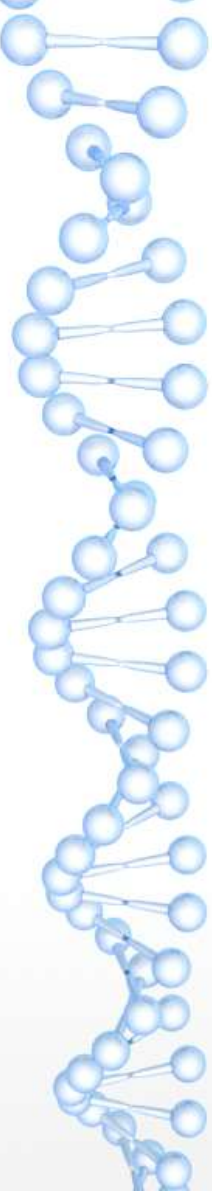
$$c[t/x] = c$$

$$(s_1 s_2)[t/x] = s_1[t/x] s_2[t/x]$$

$$(\lambda x. s)[t/x] = \lambda x. s$$

$$(\lambda y. s)[t/x] = \lambda y. (s[t/x]), \text{ если } x \neq y \text{ и либо } x \notin FV(s), \text{ либо } y \notin FV(t)$$

$$(\lambda y. s)[t/x] = \lambda z. (s[z/y][t/x]) \text{ в противном случае, причём } z \notin FV(s) \cup FV(t)$$



Свободные и связанные переменные. Подстановка и преобразования

- Альфа-преобразование: $\lambda x. s \xrightarrow{\alpha} \lambda y. s[y/x]$, при условии, что $y \notin FV(s)$. Например, $\lambda u. u v \xrightarrow{\alpha} \lambda w. w v$, но $\lambda u. u v \not\xrightarrow{\alpha} \lambda v. v v$. Такое ограничение устраняет возможность ещё одного случая захвата переменной.
- Бета-преобразование: $(\lambda x. s) t \xrightarrow{\beta} s[t/x]$.
- Эта-преобразование: $\lambda x. t x \xrightarrow{\eta} t$, если $x \notin FV(t)$. Например, $\lambda u. v u \xrightarrow{\eta} v$, но $\lambda u. u u \not\xrightarrow{\eta} u$.

Эквивалентность лямбда выражений. Экстенсинальность

$$\frac{s \xrightarrow{\alpha} t \text{ или } s \xrightarrow{\beta} t \text{ или } s \xrightarrow{\eta} t}{s = t}$$

$$\overline{t = t}$$

$$\overline{s = t}$$

$$\overline{t = s}$$

$$\overline{s = t \text{ и } t = u}$$

$$s = u$$

$$s = t$$

$$\overline{s u = t u}$$

$$\overline{s = t}$$

$$\overline{u s = u t}$$

$$s = t$$

$$\overline{\lambda x. s = \lambda x. t}$$



Эквивалентность лямбда выражений. Экстенциональность

Мы уже упоминали ранее, что η -преобразование воплощает принцип *экстенциональности*. В рамках общепринятых философских понятий два свойства называются *экстенционально эквивалентными* (либо *коэкстенсивными*), если этими свойствами обладают в точности одни и те же объекты. В теории множеств принята аксиома экстенциональности, согласно которой два множества совпадают, если они состоят из одних и тех же элементов. Аналогично, будем говорить, что две функции эквивалентны, если области их определения совпадают, а значения функций для всевозможных аргументов также одинаковы.

Введение η -преобразования делает наше понятие λ -эквивалентности экстенциональным. В самом деле, пусть $f\ x$ и $g\ x$ равны для произвольного значения x ; в частности, $f\ y = g\ y$, где переменная y выбирается так, чтобы она не была свободной как в f , так и в g . Согласно последнему из приведённых выше правил эквивалентности, $\lambda y. f\ y = \lambda y. g\ y$. Применив дважды η -преобразование, получаем, что $f = g$. С другой стороны, из экстенциональности следует, что всевозможные η -преобразования не нарушают эквивалентности, поскольку согласно правилу β -редукции $(\lambda x. t\ x)\ y = t\ y$ для произвольного y , если переменная x не является свободной в терме t . На этом мы завершаем обсуждение сущности η -преобразования и его влияния на теорию в целом, чтобы уделить больше внимания более перспективному с точки зрения вычислимости β -преобразованию.

Редукция

Отношение λ -эквивалентности, как и следовало ожидать, является симметричным. Оно достаточно хорошо соответствует интуитивному понятию эквивалентности λ -термов, но с алгоритмической точки зрения более интересен его несимметричный аналог. Определим отношение *редукции* \longrightarrow следующим образом:

$$\frac{s \xrightarrow{\alpha} t \text{ или } s \xrightarrow{\beta} t \text{ или } s \xrightarrow{\eta} t}{s \longrightarrow t}$$

$$\overline{t \longrightarrow t}$$

$$\frac{s \longrightarrow t \text{ и } t \longrightarrow u}{s \longrightarrow u}$$

$$\frac{s \longrightarrow t}{s u \longrightarrow t u}$$

$$\frac{s \longrightarrow t}{u s \longrightarrow u t}$$

$$\frac{s \longrightarrow t}{\lambda x. s \longrightarrow \lambda x. t}$$



Редукция

В действительности слово «редукция» (в частности, термин β -редукция, которым иногда называют β -преобразования) не отражает точно сути происходящего, поскольку в процессе редукции терм может увеличиваться, например:

$$\begin{aligned}(\lambda x. x x x) (\lambda x. x x x) &\longrightarrow (\lambda x. x x x) (\lambda x. x x x) (\lambda x. x x x) \\&\longrightarrow (\lambda x. x x x) (\lambda x. x x x) (\lambda x. x x x) (\lambda x. x x x) \\&\longrightarrow \dots\end{aligned}$$

Однако, несмотря на это редукция имеет прямое отношение к процедуре вычисления терма, в ходе которой последовательно вычисляются комбинации вида $f(x)$, где f — λ -абстракция. Если на некотором этапе оказывается, что не могут быть применены никакие правила редукции, кроме α -преобразований, то говорят, что терм имеет *нормальную форму*.



Редукция

$$\begin{aligned} & (\lambda x. y) ((\lambda x. x x x) (\lambda x. x x x)) \\ \longrightarrow & (\lambda x. y) ((\lambda x. x x x) (\lambda x. x x x) (\lambda x. x x x)) \\ \longrightarrow & (\lambda x. y) ((\lambda x. x x x) (\lambda x. x x x) (\lambda x. x x x) (\lambda x. x x x)) \\ \longrightarrow & \dots \end{aligned}$$

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



Редукция

Теорема 2.2 Если справедливо $s \longrightarrow t$, где терм t имеет нормальную форму, то последовательность редукций, которая начинается с терма s и состоит в применении правил редукции к самому левому редексу, всегда завершается и приводит к терму в нормальной форме.



Редукция

Пример 3.1. Рассмотрим терм $(\lambda x. xx)(\lambda x. xx)$, который сам является β -редексом, и применим к нему редукционное преобразование. $(\lambda x. xx)(\lambda x. xx) \rightarrow_{\beta} (\lambda x. xx)(\lambda x. xx) \rightarrow_{\beta} \dots$ Имеем дело с бесконечной редукционной цепочкой.

Пример 3.2. Рассмотрим терм $(\lambda x. ((\lambda y. xy)u))(\lambda v. v)$, в котором имеются два β -редекса: во-первых, сам терм является β -редексом и, во-вторых, β -редексом является его подтерм $((\lambda y. xy)u)$. Выбирая для свертки разные редексы, получим разные редукционные цепочки.

1. $(\lambda x. ((\lambda y. xy)u))(\lambda v. v) \rightarrow_{\beta} (\lambda y. (\lambda v. v)y)u \rightarrow_{\beta} (\lambda v. v)u \rightarrow_{\beta} u$
2. $(\lambda x. ((\lambda y. xy)u))(\lambda v. v) \rightarrow_{\beta} (\lambda x. (xu))(\lambda v. v) \rightarrow_{\beta} (\lambda v. v)u \rightarrow_{\beta} u$

Пример 3.3. Рассмотрим терм $(\lambda xy. y)((\lambda z. zz)(\lambda z. zz))$. Здесь два редекса: внутренний — $(\lambda z. zz)(\lambda z. zz)$ и внешний — $(\lambda x. (\lambda y. y))((\lambda z. zz)(\lambda z. zz))$. Выбрав внутренний редекс, получим его же. Таким образом, выбирая каждый раз внутренний редекс, получим бесконечную редукционную цепочку. Выбрав внешний редекс для свертывания, получим за один шаг терм без редексов. $(\lambda x. (\lambda y. y))((\lambda z. zz)(\lambda z. zz)) \rightarrow_{\beta} (\lambda y. y)$



Редукция

Заметим, что во втором случае аргумент нам редуцировать не пришлось, реализовалось так называемое ленивое вычисление.

Аппликативный порядок редукций (АПР) предписывает всегда выбирать самый левый из внутренних редексов.

Нормальный порядок редукций (НПР) предписывает всегда выбирать самый левый из внешних редексов.

Напомним, что в рассмотренном выше примере 3 первый способ выбора редекса привел к заикливанию, а второй за один шаг завершил вычисления.

Терм $\lambda x.(\lambda y.y)$ — классический пример функции, которая отбрасывает свой аргумент. Стратегия НПР откладывает вычисление аргумента до тех пор, пока он действительно не потребуется.



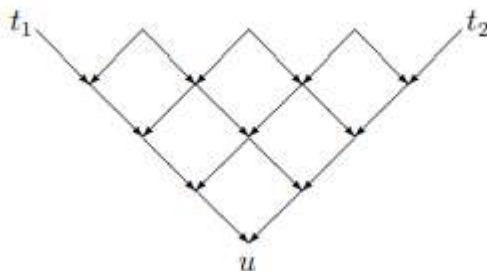
Теорема Чёрча-Россера

Теорема 2.3 Если $t \longrightarrow s_1$ и $t \longrightarrow s_2$, то существует терм u такой, что $s_1 \longrightarrow u$ и $s_2 \longrightarrow u$.

Теорема Чёрча-Россера

Corollary 2.4 Если $t_1 = t_2$ то найдётся терм u такой, что $t_1 \rightarrow u$ и $t_2 \rightarrow u$.

Доказательство: Легко показать (при помощи структурной индукции), что отношение λ -равенства = представляет собой симметричное транзитивное замыкание отношения редукции. Дальнейшее следует по индукции согласно свойствам симметричного транзитивного замыкания. Приведённая ниже диаграмма может показаться читателям, не склонным к формальным построениям, более доходчивой:



Мы полагаем, что $t_1 = t_2$, т. е. существует некоторая последовательность редукций в обоих направлениях (зигзагообразная линия в верхней части рисунка), которая их объединяет. Теорема Чёрча-Россера позволяет нам заполнить недостающие участки на краях диаграммы, после чего требуемый результат достигается композицией этих редукций.



Теорема Чёрча-Россера

Corollary 2.5 Если $t = t_1$ и $t = t_2$, причём t_1 и t_2 имеют нормальную форму, то $t_1 \equiv_{\alpha} t_2$, т. е. t_1 и t_2 равны с точностью до α -преобразований.

Доказательство: Согласно изложенному выше, найдётся некоторый терм u такой, что $t_1 \rightarrow u$ и $t_2 \rightarrow u$. Но так как t_1 и t_2 уже имеют нормальную форму, последовательность редукций, приводящая к терму u , может состоять лишь из α -преобразований. \square



Комбинаторы

$$I = \lambda x. x$$

$$K = \lambda x y. x$$

$$S = \lambda f g x. (f x)(g x)$$

Лемма 2.6 Для произвольного λ -терма t , не содержащего λ -абстракций, найдётся терм u , который также не содержит λ -абстракций и представляет собой композицию S , K , I и переменных, причём $FV(u) = FV(t) - \{x\}$ и $u = \lambda x. t$, т. е. терм u λ -равен $\lambda x. t$.

Доказательство: Применим к терму t структурную индукцию. Согласно условию, он не может быть абстракцией, поэтому нам требуется рассмотреть лишь три случая.

- Если t представляет собой переменную, возможны два случая, из которых непосредственно следует требуемый вывод: при $t = x$ мы получаем $\lambda x. x = I$, иначе, например, при $t = y$, $\lambda x. y = K y$.
- Если t — константа c , то $\lambda x. c = K c$.



Комбинаторы

- Если t представляет собой комбинацию термов, например, s и u , то согласно индуктивному предположению найдутся термы s' и u' , которые не содержат λ -абстракций и для которых справедливы равенства $s' = \lambda x. s$ и $u' = \lambda x. u$. Из этого можно сделать вывод, что $S s' u'$ является искомым выражением. В самом деле,

$$\begin{aligned} S s' u' x &= S (\lambda x. s) (\lambda x. u) x \\ &= ((\lambda x. s) x)((\lambda x. u) x) \\ &= s u \\ &= t \end{aligned}$$

Таким образом, применив η -преобразование, мы получаем $S s' u' = \lambda x. S s' u' x = \lambda x. t$, поскольку согласно индуктивному предположению переменная x не является свободной в термах s' либо u' .



Комбинаторы

Это примечательное утверждение может быть даже усилено, поскольку комбинатор I выражается через S и K . Отметим, что для произвольного A

$$\begin{aligned} S K A x &= (K x)(A x) \\ &= (\lambda y. x)(A x) \\ &= x \end{aligned}$$

Отсюда, применив η -преобразование, получаем, что $I = S K A$ для любых A . Однако, по причинам, которые станут яснее после знакомства с понятием типа, наиболее удобно положить $A = K$. Таким образом, $I = S K K$, что даёт нам возможность устранить все вхождения I в комбинаторные выражения.

Заметим, что приведённые выше доказательства имеют конструктивный характер, поскольку предлагают конкретные процедуры получения по заданному терму эквивалентного комбинаторного выражения. Процесс его построения идёт в направлении снизу вверх, и для каждой λ -абстракции, которая по построению имеет тело, свободное от λ -абстракций, применяются сверху вниз преобразования, изложенные в лемме.



Комбинаторы

Несмотря на то, что мы рассматриваем комбинаторы как некоторые термы λ -исчисления, на их основе можно сформулировать независимую теорию. Её построение начинается с определения формальных правил конструирования выражений, в которые не входит λ -абстракция, но входят комбинаторы. Далее вместо α , β и η -преобразований вводятся правила преобразования для выражений, включающих комбинаторы, например, $K\ x\ y \rightarrow x$. Такая теория будет иметь множество аналогий в традиционном λ -исчислении, например, теорема Чёрча-Россера оказывается справедливой и для приведённого выше определения редукции. Кроме того, полностью устраняются сложности со связыванием переменных. Тем не менее, мы считаем полученный формализм не слишком интуитивным, поскольку комбинаторные выражения нередко бывают весьма неясными.



Комбинаторы

Определение	Название	Характеристика
$I \equiv \lambda x. x$	тождественный	$IX \rightarrow_{\beta} X$
$B \equiv \lambda xyz. x(yz)$	компози́тор	$BFGX \rightarrow_{\beta} F(GX)$
$C \equiv \lambda xyz. xzy$	пермута́тор	$CFXY \rightarrow_{\beta} FYX$
$K \equiv \lambda xy. x$	канце́лятор	$KXY \rightarrow_{\beta} X$
$S \equiv \lambda xyz. xz(yz)$	конне́ктор	$SFGX \rightarrow_{\beta} FX(GX)$
$W \equiv \lambda xy. xyx$	дуплика́тор	$WFX \rightarrow_{\beta} FXX$

Комбинаторы

Пример 4.1.

Приведём комбинатор $B(SB(KI))(KI)(SB(SB(KI)))$ к β -нормальной форме:

$$\begin{aligned} & B(SB(KI))(KI)(SB(SB(KI))) \rightarrow_{\beta} SB(KI)(KI(SB(SB(KI)))) \\ & \rightarrow_{\beta} B(KI(SB(SB(KI)))) \left(KI(KI(SB(SB(KI)))) \right) \\ & \rightarrow_{\beta} BI \left(KI(KI(SB(SB(KI)))) \right) \rightarrow_{\beta} BII \\ & \equiv (\lambda xyz. x(yz))(\lambda x. x)(\lambda x. x) \rightarrow_{\beta} \lambda z. (\lambda x. x)((\lambda x. x)z) \\ & \rightarrow_{\beta} \lambda z. (\lambda x. x)z \rightarrow_{\beta} \lambda z. z \equiv I \end{aligned}$$