Введение в лямбда-исчисление

Косарев Дмитрий

матмех

25 марта 2024 г.

Дата сборки: 25 апреля 2024 г.

Введение: λ-исчисление

Алонзо Чёрч (1903–1995)

Алонзо Чёрч в 1935 открыл λ-исчисление

Аналогичный подход от А. Тьюринга с его машинами Тьюринга

Это разные подходы для формализации понятия «алгоритм»

В принципе, могло быть изобретено уже в 1910-х г.г.

Изображение из Википедии

Для формализации алгоритмов

λ-исчисление можно использовать как формализацию понятия «алгоритм»

Определение (Алгоритм (неформально))

Это конечная последовательность действий (или операций), к которым относятся все компьютерные программы, бюрократические процедуры, кулинарные рецепты и т.п.

Алгоритмы, которые иногда дают ответ, а иногда не завершаются, называются *разрешающими процедурами*.

Проблемы неформального определения

Зависимо от естественного языка

Не совсем понятно, что является допустимой операцией, а что нет.

- «Возьмите два любых решения уравнения $a^n + b^n = c^n$, для $n > 2$ и $a, b, c \in \mathscr{N}$ ...»
- «Если это утверждение ложно, то ...»
- «Объявим как A множество всех множеств. Если $A \in A$, то делаем одно, иначе — другое»

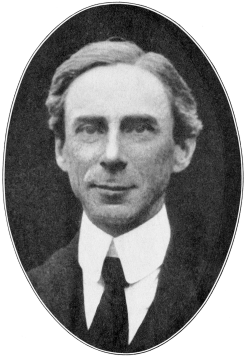
Зачем формализовывать то, что и так понятно?

«Наивная» теория множеств

Рассмотрим $P = \{y : y \notin P\}$ и задумаемся про $P \in P$?

- Если формула верна, то нарушается определение
- Если ложна, то не принадлежит, но по определению должна

Изображение из Википедии



Bertrand Russell
(1872–1970)

Цель формализации

Придумать набор недвусмысленных правил, таких что обычный офисный бюрократ (читайте, компьютер) мог им следовать и получать ожидаемый результат.

Существуют много различных формализаций:

- λ-исчисление
- Машины Тьюринга[1]
- Машины Поста
- Частично (объявленные) рекурсивные функции (англ. partial recursive function)
- Алгорифмы Маркова

[1] Мало имеют общего с компьютерами. Считать А.Тьюринга изобретателем компьютеров — неправильно

Вывод из формализации в современности

Всё, что соответствует формальному описанию алгоритма, можно запрограммировать на компьютере

Определение (Тезис Чёрча-Тьюринга)

Алгоритмом является всё то, что можно записать и исполнить в λ-исчислении (машине Тьюринга), с точностью до представления данных. И ничего более.

λ-исчисление

Процессом вычислений является переписывание программы (λ-выражения, λ-терма) на бесконечном листе бумаги.

Программы конечны и состоят из символов следующего вида.

1. Переменные, в слайдах будем их обозначать строчными латинскими буквами
2. Скобки открывающиеся (и закрывающиеся)
3. Точка как разделитель
4. Символ λ

λ-исчисление

Синтаксис:

- Переменные: $x, y, z, \ldots$
- Абстракция $(\lambda v.A)$, где A — λ-выражение, а v — произвольное имя переменной
- Применение (AB), где A и B — λ-выражения

В терминах программирования:

- Переменные
- Объявления 1-аргументных функций
- Вызов функции от одного аргумента

Каррирование

Определение (Каррирование)

представление n-арных функций через 1-арные функции

В λ-исчислении функция n аргументов представляются как функция одного аргумента, которые возвращает функцию от $n-1$ аргумента.

В мире названо в честь Хаскеля Карри. Впервые появилось в 1924 в работе М. И. Шейнфинкеля.

Моисей Исаевич Шейнфинкель (1888–1942)

Хаскел Карри (1900–1982)

Изображение взято с Википедии

Символ λ работает как квантор

$$\lambda \mathsf{m}.\lambda \mathsf{n}.\lambda \mathsf{f}.\lambda \mathsf{x}.(\mathsf{m}\ \mathsf{f}\ (\mathsf{n}\ \mathsf{f}\ \mathsf{x}))$$

- свободные вхождения
- связанных вхождения и т.д.

[1]TODO: сказать про скобочки

Подстановка

Определение (Редекс)

λ-выражение вида $(\lambda x.B)A$

Подстановка «A вместо x в выражении B» в лит-ре обозначается по-разному:

- $[x \mapsto A]B$
- $[A/x]B$

$$(\lambda x.B)A \xrightarrow{\beta} [x \mapsto A]B$$

Редекс $(\lambda x.(\lambda x.x)x)y$ вида $(\lambda v.B)A$, где

- $B \equiv (\lambda x.x)x$
- $A \equiv y$
- $v \equiv x$

$$(\lambda \mathrm{x}.(\lambda \mathrm{x}.\mathrm{x})\mathrm{x})y \to (\lambda x.x)y$$

Определение (Один шаг (β-)редукции)

это избавление от редекса $(\lambda x.B)A$ путём совершения подстановки A вместо x в выражении B

Как происходят вычисления (редукция) λ-исчислении?

Определение (Редукция)

Процесс постепенного избавления от редексов. Редукция $\equiv$ вычисление λ-выражения

Определение (Стратегия)

Порядок выбора редексов регламентирует стратегия. Ищем редексы $(\lambda x.B)A$

- Если редексов нет, то вычисление закончилось
- Если редексы есть, стратегия регламентирует какой на данном шаге редекс стоит β-редуцировать
- Или же, стратегия может сказать, что все редексы нужно оставить как есть, и выдать ответ

Что нужно для представления алгоритмов?

- Принимать входные данные
- Делать ветвления в зависимости от входных данных
- Совершать некоторое количество однотипных действий в зависимости от входных данных (т.е. должны быть циклы или их аналог — рекурсия)
  - Чтобы понимать, сколько действий уже сделали нужны натуральные числа

Историческое напоминание: числа Пеано

Первым ввёл аксиоматику арифметики в 1889 году. Натуральные числа определяются через **«базу»** и **«следующий»**

1. 0 — натуральное число
6. Для любого натурального n, $S(n)$ тоже натуральное. т.е. натуральные числа замкнуты относительно операции $S(\cdot)$
9. Аксиома индукции.

Peano's axioms in their historical context
Изображение взято с Википедии

Джузеппе Пеано (1858–1932)

Представление чисел (нумералы Чёрча). Сложение

$0 \sim (\lambda f.(\lambda x.\ x))$
$1 \sim (\lambda f.(\lambda x.\ f\ \ x))$
$2 \sim (\lambda f.(\lambda x.\ f\ \ (f\ \ x)))$
и т.д.

```
// Church numeral N
for (int i=0; i<N; i++)
  x = f(x);
```

Сложение (один из вариантов): взять два нумерала m и n, взять f и x, а затем к x применить f n раз, а затем к результату применить f m раз.

$$
\begin{aligned}
add &\equiv (\lambda m.(\lambda n.(\lambda f.(\lambda x.(m\ f\ \ (n\ f\ \ x))))))\\
&\equiv \lambda m.\lambda n.\lambda f.\lambda x.(m\ f\ \ (n\ f\ \ x))
\end{aligned}
$$

Вычисление (т.е. редукция, упрощение) 2+2

$$\Big(\lambda m.\lambda n.\lambda f.\lambda x.(m\ f\ (n\ f\ x))2\Big)2 \longrightarrow_\beta \tag{1}$$

$$\Big((\lambda m.\lambda n.\lambda f.\lambda x.(m\ f\ (n\ f\ x)))2\Big)2 \longrightarrow_\beta \tag{2}$$

$$\Big((\lambda n.\lambda f.\lambda x.(2\ f\ (n\ f\ x)))2\Big) \longrightarrow_\beta \tag{3}$$

$$\lambda f.\lambda x.(2\ f\ (2\ f\ x)) \longrightarrow \tag{4}$$

$$\lambda f.\lambda x.\Big(\Big((\lambda f.(\lambda x.f(f\ x)))f\Big)\ (2\ f\ x)\Big) \longrightarrow_\beta \tag{5}$$

$$\lambda f.\lambda x.\Big((\lambda x.f(f x))\ \ (2\ f\ x)\Big) \longrightarrow \tag{6}$$

$$\lambda f.\lambda x.((\lambda x.f(f x))\Big(\Big((\lambda f.(\lambda x.f(f\ x)))f\Big)x\Big)) \longrightarrow_\beta \tag{7}$$

$$\lambda f.\lambda x.\Big((\lambda x.f(f x))\Big((\lambda x.f(f x))x\Big)\Big) \longrightarrow_\beta \tag{8}$$

$$\lambda f.\lambda x.(\Big(\lambda x.f(f x)\Big)\Big(f(f x)\Big)) \longrightarrow_\beta \tag{9}$$

$$\lambda f.\lambda x.f(f(f(f x))) \equiv 4 \tag{10}$$

Представление чисел (нумералы Чёрча). Умножение

$0 \sim (\lambda f.(\lambda x.\ x))$
$1 \sim (\lambda f.(\lambda x.\ f\ \ x))$
$2 \sim (\lambda f.(\lambda x.\ f\ \ (f\ \ x)))$
и т.д.

```
// Church numeral N
for (int i=0; i<N; i++)
  x = f(x);
```

Умножение: взять два нумерала m и n, взять f и x, а затем к x применить n раз f, и повторить это m раз.

$$\begin{aligned} mul &\equiv (\lambda m.(\lambda n.(\lambda f.(\lambda x.((m(n\ \ f))\ x))))) \\ &\equiv \lambda m.\lambda n.\lambda f.\lambda x.((m(n\ \ f))\ x) \end{aligned}$$

Вычисление (т.е. редукция, упрощение) 2×2 длинно

$$
\begin{aligned}
((\lambda m.(\lambda n.(\lambda z.(\lambda x.(m(nz)x)))))2)2 &\longrightarrow & (1)\\
\Big((\lambda m.(\lambda n.(\lambda z.(\lambda x.(m(nz)x)))))2\Big)2 &\longrightarrow_\beta & (2)\\
\Big((\lambda n.(\lambda z.(\lambda x.(2(nz)x))))2\Big) &\longrightarrow_\beta & (3)\\
(\lambda z.(\lambda x.((2(2z))x))) &\longrightarrow & (4)\\
(\lambda z.(\lambda x.\Big(\Big((\lambda f.(\lambda x.f(f\ \ x)))(2z)\Big)x\Big))) &\longrightarrow_\beta & (5)\\
(\lambda z.(\lambda x.\Big((\lambda x.(2z(2zx)))x\Big))) &\longrightarrow_\beta & (6)\\
(\lambda z.(\lambda x.\Big((2z)(2zx)\Big))) &\longrightarrow & (7)\\
(\lambda z.(\lambda x.(\Big((\lambda f.(\lambda x.f(f\ \ x)))z\Big)(2zx)))) &\longrightarrow_\beta & (8)\\
(\lambda z.(\lambda x.((\lambda x.(z(zx)))(2zx)))) &\longrightarrow_\beta & (9)\\
(\lambda z.(\lambda x.(z(z\Big((2z)x\Big))))) &\longrightarrow & (10)\\
(\lambda z.(\lambda x.(z(z\Big(\Big((\lambda f.(\lambda x.f(f\ \ x)))z\Big)x\Big))))) &\longrightarrow_\beta & (11)\\
(\lambda z.(\lambda x.(z\Big(z\Big((\lambda x.(z(zx)))x\Big)\Big)))) &\longrightarrow_\beta & (12)\\
(\lambda z.(\lambda x.(z(z(z(zx)))))) &\equiv 4 & (13)
\end{aligned}
$$

Ветвления

$$T \equiv (\lambda x.(\lambda y.x)) \equiv fst \qquad\qquad F \equiv (\lambda x.(\lambda y.y)) \equiv snd \equiv 0$$

$$\begin{aligned}
ite &\equiv (\lambda c.(\lambda t.(\lambda e.\ ((ct)e)))) \\
(ite\ T) &\equiv (\lambda c.(\lambda t.(\lambda e.\ ((ct)e))))T \xrightarrow{\beta} \lambda t.\lambda e.((Tt)\ e) \xrightarrow{*} (\lambda t.(\lambda e.t)) \equiv T \\
(ite\ F) &\equiv (\lambda c.(\lambda t.(\lambda e.\ ((ct)e))))F \xrightarrow{\beta} \lambda t.\lambda e.((Ft)\ e) \xrightarrow{*} (\lambda t.(\lambda e.e)) \equiv F
\end{aligned}$$

Здесь $\xrightarrow{*}$ означает редукцию за несколько шагов

Рекурсия через комбинатор неподвижной точки

англ. FIXed point combinator

Не понятно как вызвать самого себя, так как имен нет.
Идея:

- Записываем функцию f так, чтобы она принимала первый аргумент, который будет вызываться вместо рекурсивного вызова
- Везде, где надо вызвать эту «рекурсивную» функцию, будем писать $Y f$

$$Y \equiv (\lambda f.(\lambda x.f\,(x\ x))(\lambda x.f\,(x\ x)))$$

Откуда такое название?

$$Y R \equiv ((\lambda f.(\lambda x.f\,(x\ x))(\lambda x.f\,(x\ x)))R) \xrightarrow{\beta} ((\lambda x.R(x\ x))(\lambda x.R(x\ x)))$$

$$\xrightarrow{\beta} R\big((\lambda x.R(x\ x))(\lambda x.R(x\ x))\big) \equiv R(Y R)$$

Получается, что $Y R$ — неподвижная точка R

Факториал с помощью Y-комбинатора (сокращённо)

$$Y\ R = R\ (Y\ R)$$

Факториал: $fac \equiv (\lambda self.(\lambda n.(\textbf{if}\ n < 2\ \textbf{then}\ 1\ \textbf{else}\ n \times self\,(n-1))))$

$$\begin{aligned}
Y\ fac\ 2 &\equiv \\
Y(\lambda self.(\lambda n.(\textbf{if}\ n < 2\ \textbf{then}\ 1\ \textbf{else}\ n \times self\,(n-1))))2 &\longrightarrow \\
(\lambda self.(\lambda n.(\textbf{if}\ n < 2\ \textbf{then}\ 1\ \textbf{else}\ n \times self\,(n-1))))(Y\,fac)2 &\longrightarrow \\
(\lambda n.(\textbf{if}\ n < 2\ \textbf{then}\ 1\ \textbf{else}\ n \times (Y\,fac)(n-1)))2 &\xrightarrow{*} \\
2 \times (Y\,fac)(2-1) &\longrightarrow_\beta \\
2 \times ((Y\,fac)1) &\xrightarrow{*} \\
2 \times (\textbf{if}\ 1 < 2\ \textbf{then}\ 1\ \textbf{else}\ n \times (Y\ fac\ (1-1))) &\xrightarrow{*} \\
2 \times 1 &\xrightarrow{*} 2
\end{aligned}$$

Ссылки

- *Practical Foundations for Programming Languages*
  https://web.archive.org/web/20210308082040/http://www.cs.cmu.edu/~rwh/pfpl/2nded.pdf
- *Slides form Harvard*
  https://groups.seas.harvard.edu/courses/cs152/2021sp/lectures/sld07-lambdacalc.pdf
- *Слайды Ю. Литвинова*
  https://github.com/yurii-litvinov/courses/tree/master/structures-and-algorithms/03-lambda-calculus
- *Peter Sestoft*
  Demonstrating Lambda Calculus Reduction

Вопросы к экзамену для ТП

1. Основные определения лямбда-исчисление. Язык, редекс, стратегия, подстановка, каррирование. Формулировка тезиса Чёрча-Тьюринга.
2. Определение и интуиция за нумералами Чёрча. Определение арифметических операций. Трассировка сложения и умножения 2 на 2 на листочке.
3. Ветвления с помощью λ-исчисления. Идея за комбинатором неподвижной точки. Набросок реализации факториала

Оглавление

Бывают различные стратегии

- Строгие (англ. strict, например, call-by-value) вычисляют аргумент до его подстановки
- Ленивые (англ. lazy, например, call-by-name) оставляют вычисление аргумента на потом

Классификация по обработке λ-абстракции

- Не вычисляют под абстракцией (например, call-by-value $\xrightarrow{cbv}$)
- Вычисляют под абстракцией (например, call-by-name $\xrightarrow{cbn}$)

На практике больше любят стратегии, которые эффективно можно посчитать

Ленивая vs. Строгая

Пример 1 ($\xrightarrow{strict}$ выглядит лучше)

$$(\lambda x.f\,xx)((\lambda x.x)A) \xrightarrow{strict} (\lambda x.f\,xx)A \xrightarrow{strict} (f\;A\;\;A) \xrightarrow{strict} \ldots$$

$$((\lambda x.f\,xx)((\lambda x.x)A)) \xrightarrow{lazy} f\Big(((\lambda x.x)A)((\lambda x.x)A)\Big) \xrightarrow{lazy} \ldots$$

Пример 2 ($\xrightarrow{lazy}$ выглядит лучше)

$$(\lambda x.(\lambda y.y))\Big((\lambda x.xx)(\lambda x.xx)\Big) \xrightarrow{strict} (\lambda x.\lambda y.y)\Big((\lambda x.xx)(\lambda x.xx)\Big) \xrightarrow{strict} \ldots \text{ зависло}$$

$$(\lambda x.(\lambda y.y))\Big((\lambda x.xx)(\lambda x.xx)\Big) \xrightarrow{lazy} (\lambda y.y) \quad \text{ответ!}$$

В обычных языках программирования: `(c>0) ? f() : g()`

Правила вывода в исчислении

Пусть дан некоторый язык L, с помощью которого записываются P_i и C.

Все $(n+1)$-местные правила вывода имеют форму:

$$\frac{P_1 \qquad \dots \qquad P_n}{C}\ \text{Название}$$

- P_i — посылки (premises)
- C_i — заключение (conclusion)

По смыслу означает «если и P_1, и P_2, ..., и P_n, то C»

Исчисление

Состоит из

- непустого множества аксиом
- множества правил вывода

Определение

Аксиома — это правило вывода без посылок. Их можно рисовать без черты

Пример исчисления. Дифференциальное исчисление

Языком L будет язык задания функций (который вообще-то надо формально определять, но не будем)

$$\frac{}{sin'(x) = cos(x)}\ \text{sin} \qquad \frac{}{cos'(x) = -sin(x)}\ \text{cos} \qquad \frac{}{x' = 1}\ \text{var} \qquad \frac{}{c' = 0, \text{если } c \in N}\ \text{const}$$

$$\frac{f\,'(x) = u(x) \qquad g'(x) = v(x)}{((f \cdot g)(x))' = u(x) \cdot g(x) + f(x) \cdot v(x)}\ \text{mul}$$

$$\frac{f'(x) = u(x) \qquad g'(x) = v(x)}{(f(g(x)))' = u(g(x)) \cdot v(x)}\ \text{cmps}$$

Дифференциальное исчисление. Пример вывода

Вывод обычно рисуется снизу вверх

$$\cfrac{\cfrac{}{\quad}\ \text{sin}\qquad \cfrac{\cfrac{}{\quad}\ \text{cos}\qquad \cfrac{\cfrac{}{\quad}\ \text{const}\quad \cfrac{}{\quad}\ \text{var}}{\quad}}{\quad}}{(sin(x) \cdot cos(2 \cdot x))' =}$$

На вывод можно смотреть как на **доказательство** того, что производная действительно посчитана правильно.
Результат вывода можно было бы упростить и дальше, но у нас недостаточно правил вывода для этого.

Дифференциальное исчисление. Пример вывода

Вывод обычно рисуется снизу вверх

$$\dfrac{\dfrac{}{sin'(x) =}\,\text{sin} \qquad \dfrac{\dfrac{}{\quad}\,\text{cos} \qquad \dfrac{\dfrac{}{\quad}\,\text{const} \quad \dfrac{}{\quad}\,\text{var}}{\quad}}{cos'(2 \cdot x) =}}{(sin(x) \cdot cos(2 \cdot x))' =}$$

На вывод можно смотреть как на **доказательство** того, что производная действительно посчитана правильно.
Результат вывода можно было бы упростить и дальше, но у нас недостаточно правил вывода для этого.

Дифференциальное исчисление. Пример вывода

Вывод обычно рисуется снизу вверх

$$\dfrac{\dfrac{}{\mathit{sin}'(x) = \mathit{cos}(x)}\,\text{sin} \qquad \dfrac{\dfrac{}{\mathit{cos}'(x) = }\,\text{cos} \qquad \dfrac{\dfrac{}{}\,\text{const} \quad \dfrac{}{}\,\text{var}}{(2 \cdot x)' = }}{\mathit{cos}'(2 \cdot x) = }}{(\mathit{sin}(x) \cdot \mathit{cos}(2 \cdot x))' = }$$

На вывод можно смотреть как на **доказательство** того, что производная действительно посчитана правильно.
Результат вывода можно было бы упростить и дальше, но у нас недостаточно правил вывода для этого.

Дифференциальное исчисление. Пример вывода

Вывод обычно рисуется снизу вверх

$$\dfrac{\dfrac{}{sin'(x) = cos(x)}\;\text{sin} \qquad \dfrac{\dfrac{}{cos'(x) = -\,sin(x)}\;\text{cos} \qquad \dfrac{\dfrac{}{2' = 0}\;\text{const} \quad \dfrac{}{x' = 1}\;\text{var}}{(2 \cdot x)' =}}{cos'(2 \cdot x) =}}{(sin(x) \cdot cos(2 \cdot x))' =}$$

На вывод можно смотреть как на **доказательство** того, что производная действительно посчитана правильно.
Результат вывода можно было бы упростить и дальше, но у нас недостаточно правил вывода для этого.

Дифференциальное исчисление. Пример вывода

Вывод обычно рисуется снизу вверх

$$\dfrac{\dfrac{}{sin'(x) = cos(x)}\,\text{sin} \qquad \dfrac{\dfrac{}{cos'(x) = -sin(x)}\,\text{cos} \qquad \dfrac{\dfrac{}{2' = 0}\,\text{const} \quad \dfrac{}{x' = 1}\,\text{var}}{(2 \cdot x)' = 0 \cdot x + 2 \cdot 1}}{cos'(2 \cdot x) =}}{(sin(x) \cdot cos(2 \cdot x))' =}$$

На вывод можно смотреть как на **доказательство** того, что производная действительно посчитана правильно.
Результат вывода можно было бы упростить и дальше, но у нас недостаточно правил вывода для этого.

Дифференциальное исчисление. Пример вывода

Вывод обычно рисуется снизу вверх

$$\dfrac{\dfrac{}{sin'(x) = cos(x)}\,\text{sin} \qquad \dfrac{\dfrac{}{cos'(x) = -sin(x)}\,\text{cos} \quad \dfrac{\dfrac{}{2' = 0}\,\text{const} \quad \dfrac{}{x' = 1}\,\text{var}}{(2 \cdot x)' = 0 \cdot x + 2 \cdot 1}}{cos'(2 \cdot x) = -sin(2 \cdot x) \cdot 2}}{(sin(x) \cdot cos(2 \cdot x))' =}$$

На вывод можно смотреть как на **доказательство** того, что производная действительно посчитана правильно.

Результат вывода можно было бы упростить и дальше, но у нас недостаточно правил вывода для этого.

Дифференциальное исчисление. Пример вывода

Вывод обычно рисуется снизу вверх

$$\dfrac{\dfrac{}{sin'(x) = cos(x)}\ \text{sin} \qquad \dfrac{\dfrac{}{cos'(x) = -\,sin(x)}\ \text{cos} \quad \dfrac{\dfrac{}{2' = 0}\ \text{const} \ \dfrac{}{x' = 1}\ \text{var}}{(2\cdot x)' = 0\cdot x + 2\cdot 1}}{cos'(2\cdot x) = -\,sin(2\cdot x)\cdot 2}}{(sin(x)\cdot cos(2\cdot x))' = cos(x)\cdot cos(2\cdot x) + sin(x)\cdot(-sin(2\cdot x)\cdot 2)}$$

На вывод можно смотреть как на **доказательство** того, что производная действительно посчитана правильно.
Результат вывода можно было бы упростить и дальше, но у нас недостаточно правил вывода для этого.

Две стратегии: Call-by-value vs. Full Reduction

$$(\lambda x.e) \to (\lambda x.e)$$

$$\frac{f \to (\lambda x.e) \qquad a \to a_2 \qquad [x \mapsto a_2]e \to r}{(f\,a) \to r}$$

$$\frac{f \to f_2 \neq (\lambda x.e) \qquad a \to a_2}{(f\,a) \to (f_2 a_2)}$$

$$v \xrightarrow{cbv} v$$

- Используется в большинстве языков программирования

$$\frac{a \to b}{(\lambda x.a) \to (\lambda x.b)}$$

$$\frac{f \to (\lambda x.e) \qquad a \to a_2 \qquad [x \mapsto a_2]e \to r}{(f\,a) \to r}$$

$$\frac{f \to f_2 \neq (\lambda x.e) \qquad a \to a_2}{(f\,a) \to (f_2 a_2)}$$

$$v \xrightarrow{full} v$$

- Считает под абстракцией, поэтому короткий ответ

λ-исчисление. Пример вывода

Вывод обычно рисуется снизу вверх

$$(\lambda x.(\lambda y.y)x)a \to$$

Отличием от дифференциального исчисления будет то, что от результата вывода надо запуститься рекурсивно пока оно не остановится. Там могло быть также, если бы были правила упрощения (например, $x - x \equiv 0$)

λ-исчисление. Пример вывода

Вывод обычно рисуется снизу вверх

$$\dfrac{\lambda x.(\lambda y.y)x \to \qquad a \to}{(\lambda x.(\lambda y.y)x)a \to}$$

Отличием от дифференциального исчисления будет то, что от результата вывода надо запуститься рекурсивно пока оно не остановится. Там могло быть также, если бы были правила упрощения (например, $x - x \equiv 0$)

λ-исчисление. Пример вывода

Вывод обычно рисуется снизу вверх

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\qquad}}{\qquad\qquad}\quad\dfrac{}{\qquad}\quad\dfrac{}{\qquad\qquad}}{(\lambda y.y)x \to}}{\lambda x.(\lambda y.y)x \to}\qquad\dfrac{}{a \to}\qquad\dfrac{}{\qquad\qquad}}{(\lambda x.(\lambda y.y)x)a \to}$$

Отличием от дифференциального исчисления будет то, что от результата вывода надо запуститься рекурсивно пока оно не остановится. Там могло быть также, если бы были правила упрощения (например, $x - x \equiv 0$)

λ-исчисление. Пример вывода

Вывод обычно рисуется снизу вверх

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{}}{(\lambda y.y) \to} \qquad \dfrac{}{x \to} \qquad \dfrac{}{}}{(\lambda y.y)x \to}}{\lambda x.(\lambda y.y)x \to} \qquad \dfrac{}{a \to} \qquad \dfrac{}{}}{(\lambda x.(\lambda y.y)x)a \to}$$

Отличием от дифференциального исчисления будет то, что от результата вывода надо запуститься рекурсивно пока оно не остановится. Там могло быть также, если бы были правила упрощения (например, $x - x \equiv 0$)

λ-исчисление. Пример вывода

Вывод обычно рисуется снизу вверх

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{y \to y}}{(\lambda y.y) \to} \qquad \dfrac{}{x \to} \qquad \dfrac{\qquad\qquad}{}}{(\lambda y.y)x \to}}{\lambda x.(\lambda y.y)x \to} \qquad \dfrac{}{a \to} \qquad \dfrac{\qquad\qquad}{}}{(\lambda x.(\lambda y.y)x)a \to}$$

Отличием от дифференциального исчисления будет то, что от результата вывода надо запуститься рекурсивно пока оно не остановится. Там могло быть также, если бы были правила упрощения (например, $x - x \equiv 0$)

λ-исчисление. Пример вывода

Вывод обычно рисуется снизу вверх

$$
\dfrac{
  \dfrac{
    \dfrac{
      \dfrac{
        \dfrac{}{y \to y}
      }{(\lambda y.y) \to (\lambda y.y)}
      \qquad
      \dfrac{}{x \to x}
      \qquad
      \dfrac{}{[x \mapsto y]y \to x}
    }{(\lambda y.y)x \to}
  }{\lambda x.(\lambda y.y)x \to}
  \qquad
  \dfrac{}{a \to}
  \qquad
  \dfrac{\qquad\qquad}{\qquad\qquad}
}{(\lambda x.(\lambda y.y)x)a \to}
$$

Отличием от дифференциального исчисления будет то, что от результата вывода надо запуститься рекурсивно пока оно не остановится. Там могло быть также, если бы были правила упрощения (например, $x - x \equiv 0$)

λ-исчисление. Пример вывода

Вывод обычно рисуется снизу вверх

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{y \to y}}{(\lambda y.y) \to (\lambda y.y)} \qquad \dfrac{}{x \to x} \qquad \dfrac{}{[x \mapsto y]y \to x}}{(\lambda y.y)x \to x}}{\lambda x.(\lambda y.y)x \to} \qquad \dfrac{}{a \to} \qquad \dfrac{}{\qquad\qquad}}{(\lambda x.(\lambda y.y)x)a \to}$$

Отличием от дифференциального исчисления будет то, что от результата вывода надо запуститься рекурсивно пока оно не остановится. Там могло быть также, если бы были правила упрощения (например, $x - x \equiv 0$)

λ-исчисление. Пример вывода

Вывод обычно рисуется снизу вверх

$$\dfrac{\dfrac{\dfrac{\dfrac{}{y \to y}}{(\lambda y.y) \to (\lambda y.y)} \qquad \dfrac{}{x \to x} \qquad \dfrac{}{[x \mapsto y]y \to x}}{(\lambda y.y)x \to x}}{\lambda x.(\lambda y.y)x \to (\lambda x.x)} \qquad \dfrac{}{a \to} \qquad \dfrac{}{}}{(\lambda x.(\lambda y.y)x)a \to}$$

Отличием от дифференциального исчисления будет то, что от результата вывода надо запуститься рекурсивно пока оно не остановится. Там могло быть также, если бы были правила упрощения (например, $x - x \equiv 0$)

λ-исчисление. Пример вывода

Вывод обычно рисуется снизу вверх

$$\frac{\dfrac{\dfrac{\dfrac{\dfrac{}{y \to y}}{(\lambda y.y) \to (\lambda y.y)} \qquad \dfrac{}{x \to x} \qquad \dfrac{}{[x \mapsto y]y \to x}}{(\lambda y.y)x \to x}}{\lambda x.(\lambda y.y)x \to (\lambda x.x)} \qquad \dfrac{}{a \to a} \qquad \dfrac{}{[a \mapsto x]x \to}}{(\lambda x.(\lambda y.y)x)a \to}$$

Отличием от дифференциального исчисления будет то, что от результата вывода надо запуститься рекурсивно пока оно не остановится. Там могло быть также, если бы были правила упрощения (например, $x - x \equiv 0$)

λ-исчисление. Пример вывода

Вывод обычно рисуется снизу вверх

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{y \to y}}{(\lambda y.y) \to (\lambda y.y)} \qquad \dfrac{}{x \to x} \qquad \dfrac{}{[x \mapsto y]y \to x}}{(\lambda y.y)x \to x}}{\lambda x.(\lambda y.y)x \to (\lambda x.x)} \qquad \dfrac{}{a \to a} \qquad \dfrac{}{[a \mapsto x]x \to a}}{(\lambda x.(\lambda y.y)x)a \to}$$

Отличием от дифференциального исчисления будет то, что от результата вывода надо запуститься рекурсивно пока оно не остановится. Там могло быть также, если бы были правила упрощения (например, $x - x \equiv 0$)

λ-исчисление. Пример вывода

Вывод обычно рисуется снизу вверх

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{y \to y}}{(\lambda y.y) \to (\lambda y.y)} \qquad \dfrac{}{x \to x} \qquad \dfrac{}{[x \mapsto y]y \to x}}{(\lambda y.y)x \to x}}{\lambda x.(\lambda y.y)x \to (\lambda x.x)} \qquad \dfrac{}{a \to a} \qquad \dfrac{}{[a \mapsto x]x \to a}}{(\lambda x.(\lambda y.y)x)a \to a}$$

Отличием от дифференциального исчисления будет то, что от результата вывода надо запуститься рекурсивно пока оно не остановится. Там могло быть также, если бы были правила упрощения (например, $x - x \equiv 0$)