Лямбда-исчисление

Косарев Дмитрий

матмех

20 марта 2024 г.

Дата сборки: 18 марта 2024 г.

Введение: λ-исчисление

Alonzo Church (1903–1995)

Алонзо Чёрч 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*.

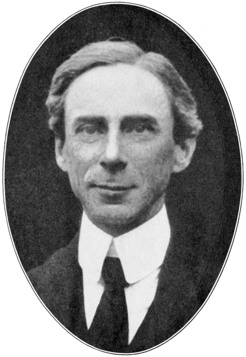
Проблемы неформального определения

- 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 $a, b, c \in \mathcal{N}$ ...»
  - «Если это утверждение ложно, то ...»
  - «Объявим как A множество всех множеств. Если $A \in A$, то делаем одно, иначе — другое»

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

Историческое напоминание: числа Пеано

Первым ввел 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

$0 \sim (\lambda f.(\lambda x.\ x))$
$1 \sim (\lambda f.(\lambda x.\ s\ x))$
$2 \sim (\lambda f.(\lambda x.\ s\ (s\ x)))$
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.

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

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.

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

Символ λ работает как квантор

$$\lambda m.\lambda n.\lambda f.\lambda x.(m\ f\ (n\ f\ x))$$

- свободные вхождения
- связанных вхождения и т.д.

[1]TODO: сказать про скобочки

Подстановка

Определение

Редекс — это 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

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

$$(\lambda m.\lambda n.\lambda f.\lambda x.(m\ f\ (n\ f\ x)))22 \longrightarrow_\beta$$

$$(\lambda m.\lambda n.\lambda f.\lambda x.(m\ f\ (n\ f\ x)))22 \longrightarrow_\beta$$

$$(\lambda n.\lambda f.\lambda x.(2\ f\ (n\ f\ x)))2 \longrightarrow_\beta$$

$$\lambda f.\lambda x.(2\ f\ (2\ f\ x)) \longrightarrow$$

$$\lambda f.\lambda x.((\lambda f.(\lambda x.f(f x)))\ f\ (2\ f\ x)) \longrightarrow_\beta$$

$$\lambda f.\lambda x.((\lambda x.f(f x))\ \ (2\ f\ x)) \longrightarrow$$

$$\lambda f.\lambda x.((\lambda x.f(f x))\ \ (((\lambda f.(\lambda x.f(f x)))\ f\ \ x))) \longrightarrow_\beta$$

$$\lambda f.\lambda x.((\lambda x.f(f x))((\lambda x.f(f x))x)) \longrightarrow_\beta$$

$$\lambda f.\lambda x.((\lambda x.f(f x))(f(f x))) \longrightarrow_\beta$$

$$\lambda f.\lambda x.f(f(f(f x))) \equiv 4$$

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

$$
\begin{aligned}
(\lambda m.(\lambda n.(\lambda z.(m(nz)))))22 &\longrightarrow \\
(\lambda m.(\lambda n.(\lambda z.(m(nz)))))22 &\longrightarrow_\beta \\
(\lambda n.(\lambda z.(2(nz))))2 &\longrightarrow_\beta \\
(\lambda z.2(2z)) &\longrightarrow_\beta \\
(\lambda zx.(2z(2zx))) &\longrightarrow_\beta \\
(\lambda zx.(\lambda x.(z(zx)))(2zx)) &\longrightarrow_\beta \\
(\lambda zx.(z(z(2zx)))) &\longrightarrow_\beta \\
(\lambda zx.(z(z(\lambda x.(z(zx)))x))) &\longrightarrow_\beta \\
(\lambda zx.(z(z(z(zx))))) &\equiv 4
\end{aligned}
$$

Определения алгоритма

Теорема (Тезис Чёрча)

Используя λ-исчисление можно реализовать произвольный алгоритм (с точностью до представления данных).

Теорема (Тезис Тьюринга)

Используя машину Тьюринга можно реализовать произвольный алгоритм (с точностью до представления данных).

Т.е. теперь под алгоритмом понимается только то, что можно записать в формализме (-ах).

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

Ветвления

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

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

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?

$$YR = (\lambda x.R(x\ x))(\lambda x.R(x\ x)) \to R\Big((\lambda x.R(x\ x))(\lambda x.R(x\ x))\Big) = R(YR)$$

Получается, что YR — неподвижная точка R

Факториал с помощью комбинатора неподвижной точки (сокращённо)

$$\mathtt{FIX}\ R = R\ (\mathtt{FIX}\ R)$$

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

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

Ссылки

Слайды Ю. Литвинова
https://github.com/yurii-litvinov/courses/tree/master/structures-and-algorithms/03-lambda-calculus

Оглавление

Бывают различные стратегии

Например,

- Строгие (например, call-by-value $\xrightarrow{cbv}$) вычисляют аргумент до его подстановки
- Ленивые (например, call-by-name $\xrightarrow{cbn}$) оставляют вычисление аргумента на потом

На практике больше любят стратегии, которые эффективно можно посчитать

Ленивая vs. Строгая

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

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

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

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

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

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

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