λ Calculus

WilliamLiusy

2024/7/11

Introduction

- Hilbert's 10th problem
- Turing Machine, λ Calculus

Introduction

- Hilbert's 10th problem
- Turing Machine, λ Calculus
- Calculus: A set of rules to formally manipulate symbols.

Introduction

- Hilbert's 10th problem
- Turing Machine, λ Calculus
- Calculus: A set of rules to formally manipulate symbols.

λ Calculus is more concise, more elegant, more abstract.

Alonzo Church

Definition

- $< Expression >:=< Name > / < Function > / < Application >$
- $< Function >:= (\lambda < Name > . < Expression >)$
- $< Application >:=< Expression > < Expression >$

e.g.

- $(\lambda x.x)(\lambda x.x)$
- $(\lambda x.(\lambda y.(+\ x\ y))\ 3)\ 2$ (IF we temporarily consider "+" as an inbuilt function)

Calculus Rules

- α-conversion: Substitution in Mathematical Logic
- β-reduction: Evaluation

Currying

Note that we only allow one function to own **ONE** parameter.
How to pass more parameters?

Currying

Note that we only allow one function to own **ONE** parameter.
How to pass more parameters?

Recall: $(\lambda x.(\lambda y.(+\ x\ y))\ 3)\ 2$
Here, $(\lambda y.(+\ x\ y))\ 3)$ is an application.
After evaluation, it returns $(+\ x\ 3)$.
Combined with the outer λ, we get a function$(\lambda x.(+\ x\ 3))$

Currying

Note that we only allow one function to own **ONE** parameter.
How to pass more parameters?

Recall: $(\lambda x.(\lambda y.(+\ x\ y))\ 3)\ 2$
Here, $(\lambda y.(+\ x\ y))\ 3)$ is an application.
After evaluation, it returns $(+\ x\ 3)$.
Combined with the outer λ, we get a function$(\lambda x.(+\ x\ 3))$

Haskell Function Declaration:
sum::int->int->int
sum x y = x + y

Evaluation Order

Again: $(\lambda x.(\lambda y.(+\ x\ y))\ 3)\ 2$
We can evaluate it in two ways:
(1) Outer-to-Inner:
$(\lambda x.(\lambda y.(+\ x\ y))\ 3)\ 2 \quad \Rightarrow \quad (\lambda y.(+\ 2\ y))\ 3) \quad \Rightarrow \quad (+\ \ 2\ \ 3)$
(2) Inner-to-Outer:
$(\lambda x.(\lambda y.(+\ x\ y))\ 3)\ 2 \quad \Rightarrow \quad (\lambda x.(+\ x\ 3))\ 2 \quad \Rightarrow \quad (+\ \ 2\ \ 3)$

Evaluation Order

Again: $(\lambda x.(\lambda y.(+\ x\ y))\ 3)\ 2$
We can evaluate it in two ways:
(1) Outer-to-Inner:
$(\lambda x.(\lambda y.(+\ x\ y))\ 3)\ 2 \quad \Rightarrow \quad (\lambda y.(+\ 2\ y))\ 3) \quad \Rightarrow \quad (+\ \ 2\ \ 3)$
(2) Inner-to-Outer:
$(\lambda x.(\lambda y.(+\ x\ y))\ 3)\ 2 \quad \Rightarrow \quad (\lambda x.(+\ x\ 3))\ 2 \quad \Rightarrow \quad (+\ \ 2\ \ 3)$

However,$(\lambda x.1)\ (\lambda x.x)\ (\lambda x.x)$
(1) Left-to-Right: 1
(2) Right-to Left: Never Terminates.

Theorem (Church-Rosser)

An expression cannot be evaluated into two distinct normal form.

Theorem (Church-Rosser)

If an expression can be converted into a normal form, it can always be converted into it in the normal order.

So, now we always compute in the normal order. This will be important when it comes to recursion.

Arithmetics

Now back to the pure theory. Our world now only has expressions. But we need at least Arithmetic Structures of Natural Numbers in programming.

Arithmetics

Now back to the pure theory. Our world now only has expressions. But we need at least Arithmetic Structures of Natural Numbers in programming.

- $0 := (\lambda f, x.\ x)$
- $1 := (\lambda f, x.\ f\, x)$
- $2 := (\lambda f, x.\ f\, (f\, x))$

  .....

Ring Structure

(1) Addition:
Denote S as $(\lambda n, a, b.\ a\ (n\ ab))$ (The Successor)

Ring Structure

(1) Addition:
Denote S as $(\lambda n, a, b.\ a\ (n\ ab))$ (The Successor)
$a + b := aSb$

Ring Structure

(1) Addition:
Denote S as $(\lambda n, a, b.\ a\ (n\ ab))$ (The Successor)
$a + b := aSb$
(2) Multiplication:
Denote M as $(\lambda x, y, a.\ x\ (y\ a))$
$a \times b := M\ a, b$

Logic

(1) True/False:
Denote T as $(\lambda x, y.x)$
Denote F as $(\lambda x, y.y)$
(2) Logical Operands:
Denote $\wedge$ as $(\lambda x, y.xyF)$
Denote $\vee$ as $(\lambda x, y.xTy)$
Denote $\neg$ as $(\lambda x.xFT)$
(3) If Clauses: Denote Z as $(\lambda x.xF\neg F)$
$Zn = T$ if $n \neq 0$
$Zn = F$ if $n = 0$

Equality

(1) Predecessor:
A pair (a,b) is defined by: $(\lambda z.z\ ab)$
Denote Φ as $(\lambda p, z.z\ (S\ (p\ T))(p\ T))$
Denote P as $(\lambda n.(n\Phi\ (\lambda z.z\ 00))\ F)$ (The Predecessor)
(2) Equality and Inequality

Recursion

Denote Y as $(\lambda y.(\lambda x.y\,(x\,x))\;(\lambda x.y\,(x\,x)))$
It is called the Y combinator.
When it receives a function R:

$$\begin{aligned} Y\,R &= (\lambda x.R\,(x\,x))\;(\lambda x.R\,(x\,x)) \\ &= R\,(\lambda x.R\,(x\,x))\;(\lambda x.R\,(x\,x)) \\ &= R\,(Y\,R) \end{aligned}$$

For example, we compute the Factorial of n.
Take R as $(\lambda FACT, n.Z\;n\;1(M\;n(FACT(P\;n))))$
Then Y R returns the answer.

Turing Completeness

- God creates natural numbers, and then human create everything.

Turing Completeness

- God creates natural numbers, and then human create everything.
- Church-Turing Thesis

Reference

- A Tutorial Introduction to the Lambda Calculus, Raul Rojas
- https://www.bilibili.com/video/BV1sZ4y1n7oz/?spm_id_from=333.337.search-card.all.click&vd_source=2b5cec79fa7ea88fa0bc7574ca679b15
- Kimi.ai