Lambda Calculus
Introduction and completeness

27 de enero de 2016

Haritz Puerto-San-Román
hpuerto@uma.es
Felipe Sulser-Larraz
felipesulser@gmail.com

ETSI Informática
Universidad de Málaga
España

UNIVERSIDAD
DE MÁLAGA

Agenda

History
Syntax
Beta-conversion
Alfa-conversion
Combinators
Beta normal form
Examples
Data Structures
Church numerals
Eta-Conversion
Fixed Points
Decision Problem
Turing Completeness
Bibliography

Quote

"I don't believe in empirical science. I only believe in a priori truth"

Kurt Gödel

History

- λ calculus invented by Church in 1928 and first published in 1932.
- Formal system:
  - Designed to investigate functions and recursion (foundations of mathematics)
  - The original system was shown to be logically inconsistent in 1935 by Stephen Kleene and J. B. Rosser who developed the Kleene–Rosser paradox.
  - In 1936 Church published just the portion relevant to computation, what is now called the untyped lambda calculus.
  - In 1940, he also introduced a computationally weaker, but logically consistent system, known as the simply typed lambda calculus.

Syntax

$$e ::= x \mid \lambda x.e \mid ee$$

Being:

- x a variable
- λ x . e is a λ abstraction (function). x is the argument and e is the body.
- ee is a λ application.

We call the set of all λ-terms Λ

Examples

Which of these expressions are right?

- λ (x.x)
- (λ(x. y)
- λx.(xy)
- (λx.x)y

Syntax

$$(\underbrace{\lambda y}_{\text{head}}.\underbrace{x(yz)}_{\text{body: an expression}})\ \underbrace{(ab)}_{\text{another expression, after the function}}$$

Beta-conversion

$$(\lambda x.e1)e2- > e1[x/e2]$$

This reduction describes the function application rule.
e2 is passed to the function.
Example:
Assume sqr and 3 are defined:

- $(((\lambda f.(\lambda x.(f(f(x))))sqr)3)$
- $((\lambda x.(sqr(sqr(x)))3)$
- $(sqr(sqr\ 3))$
- $(sqr\ 9)$
- 81

Examples

A simple β reduction

- $(\lambda x.(\lambda z(xz)))y$
- $\lambda z.(yz)$

A currified function

- $(\lambda x.\lambda y.xy)z$
- $\lambda y.zy \rightarrow$ this is like currying in haskell!

ETSI Informática
Universidad de Málaga
España

Static Scope

λ-calculus uses static scoping:

Question:
Does $(\lambda x.x(\lambda x.x))z$ equals to $(\lambda x.x(\lambda y.y))z$?

Yes, both x's are bound to a λ. x can be anything.

Alfa conversion: $\lambda x.x = \lambda y.y$

α conversion

ETSI Informática
Universidad de Málaga
España

This is simply renaming a bound variable.

$\lambda v.E \rightarrow \lambda w.E[v \rightarrow w]$

Examples

$\lambda y.(\lambda f.fx)y \xrightarrow{a} \lambda z.(\lambda f.fx)z \xrightarrow{a} \lambda z.(\lambda g.gx)z$

Conventions

Looking at λ terms can be very hard to decipher, so we will omit outer parenthesis whenever possible and we will use association to the left.

- $(\lambda x.(\lambda y.yx)) = \lambda x.\lambda y.yx$

And instead of that, we will write:

- $\lambda xy.yx$

Example

$(\lambda x.xy)(\lambda z.z)w = (\lambda z.z)yw = yw$

Combinators

A λ-term M is a called a combinator if $FV(M) = \varnothing$.

Example

- $I = \lambda x.x$
- $K = \lambda xy.x$
- $S = \lambda xyz.xz(yz)$
- $\omega = \lambda x.xx$
- $\Omega = \omega\omega$
- $Y = \lambda f.(\omega(\lambda x.f(xx)))$

β normal form

Haritz Puerto-San-Roman and Felipe Sulser-Larraz

We say that a term λ is in β normal form if it cannot be β-reduced. A term has a β normal form if it β reduces to a term that has a β normal form.

Example

I is in β-nf. Ω does not have a β-nf.
$KI\Omega$ not in β-nf but it has one, namely I

ETSI Informática
Universidad de Málaga
España

Examples - Truth values

Let's define true and false values:

- T = $\lambda tf.t$
- F = $\lambda tf.f$

T is a function that takes 2 arguments and returns the first one
F is a function that takes 2 arguments and returns the last one

Examples - Truth values

Let's see an example of T and F

- if T then e1 else e2 = T e1 e2 = $(\lambda tf.t)$ e1 e2 = e1
- if F then e1 else e2 = F e1 e2 = $(\lambda tf.f)$ e1 e2 = e2

Examples - Definition of Logical gates

- AND = λ xy . xyF
- OR = λ xy . xTy
- NOT = λ x . xFT

ETSI Informática
Universidad de Málaga
España

Assume e1 = T

- AND e1 e2 => e1 e2 F => T e2 F => e2
- OR e1 e2 => e1 T e2 => T T e2 => T
- NOT e1 => e1 F T => T F T => F

Assume e1 = F

- AND e1e2 => e1 e2 F => F e2 F => F
- OR e1 e2 => e1 T e2 => F T e2 => e2
- NOT e1 => e1 F T => F F T = T

Data Structures - pair

How can we represent a tuple?

- pair = λxyb . b x y
- fst = λp. p T
- snd = λp. p F

Data Structures - examples

- fst = λp. p T
- snd = λp. p F
- fst (pair e1 e2) => (pair e1 e2) T => (λb . b e1 e2) T => T e1 e2 => e1

Church numerals

There are many ways to represent numbers using λ calculus. We will use the following:

- $0 = \lambda \, f \, x \, . \, x$

Church numerals

ETSI Informática
Universidad de Málaga
España

There are many ways to represent numbers using λ calculus. We will use the following:

- 0 = λ f x . x
- 1 = λ f x . fx

Church numerals

There are many ways to represent numbers using λ calculus. We will use the following:

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

How can we obtain the successor of a number?

SUCC := λ nfx.f (n f x)

Let's try it out:

- SUCC 0 = SUCC (λ f x. x) = (λn f x. f (n f x)) (λf x. x) = λf x . f ((λf x. x) f x)) = λ f x . f (x) = 1

ETSI Informática
Universidad de Málaga
España

How can we obtain the successor of a number?

SUCC := λ nfx.f (n f x)

Let's try it out:

- SUCC 0 = SUCC (λ f x. x) = (λn f x. f (n f x)) (λf x. x) = λf x . f ((λf x. x) f x)) = λ f x . f (x) = 1
- SUCC 1 = SUCC (λfx.fx) = (λn f x. f (n f x)) (λf x.fx) => λf x . f((λf x.fx) f x) = λf x.f (f x) = 2

Lambda Calculus

Haritz Puerto-San-Roman and Felipe Sulser-Larraz

ETSI Informática
Universidad de Málaga
España

Predecessor function

We need the auxiliary function next.

- next = λp . pair (snd p) (add (snd p) 1)
- next (pair a b) = pair b (b+1)
- It can be shown that by applying next to pair 0 0 exactly n times, we obtain pair (n-1) n

PRED := λn . fst ($next^n$ (pair 0 0))

PRED 1 = (λn . fst ($next^n$ (pair 0 0))) 1 = fst (pair 0 1) = 0

ETSI Informática
Universidad de Málaga
España

Zero? function

λnxy . n (λz.y) x

Example

- Zero? 0 = (λnxy . n (λz.y) x) (λfx . x) = λxy. (λfx. x) (λz.y) x = λxy. x = T
- Zero? 1 = (λnxy . n (λz.y) x) (λfx . fx) = λxy. (λfx. fx) (λz.y) x = λxy. (λz.y) x = λxy. y = F

η-Conversion

ETSI Informática
Universidad de Málaga
España

Let's see it with an example:
If x does not appear in f, then
$(\lambda\ x.\ f\ x)\ g = f\ g$

Pointfree Programming

It is very common for functional programmers to write functions as a composition of other functions, never mentioning the actual arguments they will be applied to. For example, compare:

sum = foldr (+) 0

with:

sum' xs = foldr (+) 0 xs

Fixed Points Exist

For every $M \in \Lambda$ there exists $X \in \Lambda$ such that $M X = X$, that is X is a fixed point of M .

Proof

We claim that YM is a fixed point of M.

$YM = (\lambda x \,.\, M\,(xx))(\lambda x \,.\, M\,(xx))$

$= M\,((\lambda x \,.\, M\,(xx))(\lambda x \,.\, M\,(xx)))$

$= M\,(YM\,)$

Fixed Points - Application

ETSI Informática
Universidad de Málaga
España

Let's see an application.

$$add\ n\ m = \begin{cases} m & if \quad n = 0 \\ add(n-1)(m+1) & otherwise \end{cases}$$

ADD = λxy. (Zero? x) (y) (ADD (Pred x) (Succ y))

There is a problem here. In the definition of add we are referencing add so let's abstract out add.

New definition of ADD

ADD = λpxy. (Zero? x) (y) (p (Pred x) (Succ y)) ADD
Q = λpxy. (Zero? x) (y) (p (Pred x) (Succ y))
ADD is a fixed point of Q
ADD = YQ

Now, ADD is not used in its definition.

Let's check its behaviour

ADD n m = YQ n m = Q(YQ) n m = Q (ADD) n m = (Zero? n) (m) (ADD (Pred n) (Succ m))

Godel Numbering

There exists an effect enumeration of λ-terms. For M $\in \lambda$ we write #M to denote the Godel number of M . We write [#M] to stand for the λ-term representing #M .

Theorem

For every F $\in \Lambda$ there is an X $\in \Lambda$ such that F [#X] = X.

Proof

All recursive functions are λ-definable by the Church-Turing Thesis. By the effectiveness of our numbering, there is a term N such that:

N[#M] = [#[#M]]

Furthermore, there is a term A such that

A[#M][#N] = [#(M N)]

Now, let's take W = λn. F(An (N n))

X = W[#W] = F(A [#W](N[#W])) = F(A[#W]([#[#M]])) = F([#(W[#W])]) = F([#X])

Alonzo Church proved that there is no term that decides whether two terms have the same normal form.
He reduced this problem to asking whether a given term has a normal form, and then showed this problem can't be answered using a λ-term. We will only show this proof

Theorem

There is no lambda term, M, such that

$$M\ n = \begin{cases} 0 & \textit{if term with Godel number } n \textit{ has a } \beta nf \\ 1 & \textit{otherwise} \end{cases}$$

Proof

Let's suppose there is such M.

Let's define G = λ n. Zero?(M n) Ω I

As we shown before, there is an X such that:

G[#X] = X

Let's suppose x has a β-nf.

M[#X] = 0 => G[#X] = Zero? (0) ΩI = Ω = X => X has no β-nf.

CONTRADICTION!!

ETSI Informática
Universidad de Málaga
España

Proof (2^{nd} part)

Let's suppose x has no a β-nf.
M[#X] = 1 => G[#X] = Zero? (1) ΩI = I = X => X has β-nf.
CONTRADICTION!!
We can conclude that there is no such M.

We will show the equivalence to μ-recursive functions.

μ recursive functions

1. Constant function: $f(x_1, ...x_k) = n$
2. Successor function: $S(x) = f(x) = x + 1$
3. Projection function:$P(i, k) = f(x_1, ...x_k) = x_i$

Operators:

1. Composition operator
2. Primitive recursion
3. Minimalisation

Constant function and Successor function straightforward.
And the Projection function?

Projection: $f(x_1, ..., x_k) = x_i$
In lambda terms:
$\lambda x_1, ..., x_k.x_i$

Composition

Definition 9.5: *Composition operation*

Given $m > 0$, $k \geq 0$ and the functions:

$$g : \mathbf{N}^m \to \mathbf{N}$$

$$h_1, h_2, \ldots, h_m : \mathbf{N}^k \to \mathbf{N}$$

If the Function $f : \mathrm{N}^k \to \mathrm{N}$ is:

$$f(\underline{n}) = g(\, h_1(\underline{n}) \,,\, h_2(\underline{n}) \,, \ldots,\, h_m(\underline{n}) \,)$$

then we say **that** f **is the composition of** g ***with*** $h_1, h_2, \ldots, h_m$.

We will denote $f(\underline{n}) = g(h_1, h_2, \ldots, h_m)(\underline{n})$, or simply $f = g(h_1, h_2, \ldots, h_m)$.

ETSI Informática
Universidad de Málaga
España

In terms of lambda calculus:
$\lambda gh1h2...hmn1n2...nk.$
$g(h1\ n1\ n2...nk)(h2\ n1....nk)...(hm\ n1\ n2...nk)$

The spaces are added for the purpose of understanding the formulae.

Definition 9.6: *Recursively defined function*

Given $k \geq 0$, and the functions:

$$g : \mathrm{N}^k \longrightarrow \mathrm{N}$$
$$h : \mathrm{N}^{k+2} \longrightarrow \mathrm{N}$$

The Function $f : \mathrm{N}^{k+1} \rightarrow \mathrm{N}$ such that is defined as:

$$f(\underline{n},m) = \begin{cases} g(\underline{n}) & \text{if } m = 0 \\ h(\underline{n},m-1,f(\underline{n},m-1)) & \text{if } m > 0 \end{cases}$$

we say that f **is defined recursively by** g **and** h.

We will denote it as $f(\underline{x}) = \langle g|h \rangle(\underline{x})$, or simply $f = \langle g|h \rangle$.

Primitive Recursion in Lambda

Lambda Calculus

Haritz Puerto-San-Roman and Felipe Sulser-Larraz

ETSI Informática
Universidad de Málaga
España

We will use the previously explained Y combinator.

$\lambda g\ h\ n1\ \ n2 ... nk.$
$Y(\lambda fm.iszero\ m(f\ \ n1\ \ n2 ... nk)$
$(g\ n1\ \ n2 ... nk)(prec\ m)(f(prec\ m))))$

Definition 9.7: *Unbounded Minimalization*

Given $k \geq 0$ and the Function:
$g : \mathrm{N}^{k+1} \rightarrow \mathrm{N}$

If the Function $f : \mathrm{N}^{k} \rightarrow \mathrm{N}$ is:

$$f(\underline{n}) = \begin{cases} minimum(\mathrm{A}) & if\ A \neq \varnothing \quad \wedge \quad \forall t \leq minimum(\mathrm{A}) \quad g(\underline{n},t) \in \mathrm{N} \\ \uparrow & \text{otherwise} \end{cases}$$

where $A = \{\, t \in \mathrm{N} \;/\; g(\underline{n},t) = 0 \,\}$ and $\underline{n} \in \mathrm{N}^{k}$

then we say that f is obtained from g by unbounded minimalization.
We will denote it as $f(\underline{n}) = \mu[g](\underline{n})$, or simply $f = \mu[g]$.

Note: The symbol "$\uparrow$" means that the Function, for that input vector ($\underline{n}$), verifies that: $\underline{n} \notin Dom(f)$. That is, the Function diverges ("it is not defined") for that input .

Minimalisation in Lambda

Haritz Puerto-San-Roman and Felipe Sulser-Larraz

Again, we'll use the Y combinator.

$\lambda g\ n1\ n2 ... nk.$

$(Y.(\lambda h\ x.zero?(g\ x1\ x2...xk\ x)x(h(succ\ x)))zero)$

ETSI Informática
Universidad de Málaga
España

Conclusion

We have provided:

1. Syntax definition
2. Rules of derivation and conversion
3. Simple data structures and Church's encoding
4. Recursion in Lambda Calculus
5. Decision Problem in Lambda Calculus
6. Equivalence for the Turing completeness in Lambda Calculus

Bibliography

[1] Gunther 2010, *Lambda Calculus and the Decision Problem*

[2] Roger Hindley and Jonathan Seldin, *Introduction to Combinators and Lambda calculus*

[3] Sungwoo Park (POSTECH), *CSE-321 Programming Languages, Untyped Lambda Calculus*

[4] Raul Rojas (FU Berlin), *A Tutorial Introduction to the Lambda Calculus*

[5] Ramos-Jimenez, G. (UMA) *Automata Theory and Formal Languages*