Lambda Calculus

Diogo Sousa

Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia

January 27, 2012

What is Lambda Calculus?

- Turing-complete computational model.
- Simple grammar and axioms.
- Based on untyped, anonymous functions.
  - Foundation for functional programming languages (extended with types)

History of Lambda Calculus

- Invented by Alonzo Church in 1928 (published in 1932), eight years before the Turing machine (1936).
- Aimed to be a formal system for the foundations of logic (before Gödel's incompleteness theorem in 1931).
- Found logically inconsistent in 1933 by Stephen Kleene and Barkley Rosser, both Church students.

Alonzo Church
(1903–1995)

Grammar of Lambda Calculus

Definition (BNF Grammar)

$\langle E \rangle$ ::=	v	variable
|	<E> <E>	application
|	λ v . <E>	abstraction

Grammar of Lambda Calculus

Definition (BNF Grammar)

$\langle E \rangle$ ::=	v	variable
|	<E> <E>	application
|	λ v . <E>	abstraction

Definition (λ-terms)

Let V be the set of variables identifiers, then we define Λ, the set of λ-terms as

1. $x \in V \implies x \in \Lambda$ (variables)
2. $M, N \in \Lambda \implies M\,N \in \Lambda$ (applications)
3. $M \in \Lambda \implies \lambda v.M \in \Lambda$ (abstractions)
4. Nothing more belongs to Λ.

Substitution

Definition (Substitution)

Let E be a λ-term, then $E[x \mapsto r]$ denotes the substitution of x for r in E.

More formally, we define substitution inductively on the structure of λ-terms:

1. $x[x \mapsto r] = r$;
2. $y[x \mapsto r] = y$ if $y \neq x$;

Substitution

Definition (Substitution)

Let E be a λ-term, then $E[x \mapsto r]$ denotes the substitution of x for r in E.

More formally, we define substitution inductively on the structure of λ-terms:

1. $x[x \mapsto r] = r$;
2. $y[x \mapsto r] = y$ if $y \neq x$;
3. $(t\ s)[x \mapsto r] = t[x \mapsto r]\ s[x \mapsto r]$;

Substitution

Definition (Substitution)

Let E be a λ-term, then $E[x \mapsto r]$ denotes the substitution of x for r in E.

More formally, we define substitution inductively on the structure of λ-terms:

1. $x[x \mapsto r] = r$;
2. $y[x \mapsto r] = y$ if $y \neq x$;
3. $(t\ s)[x \mapsto r] = t[x \mapsto r]\ s[x \mapsto r]$;
4. $(\lambda x.t)[x \mapsto r] = \lambda x.t$;
5. $(\lambda y.t)[x \mapsto r] = \lambda y.(t[x \mapsto r])$ if $y \neq x$ and $\underbrace{\text{y is fresh for r}}_{\text{otherwise needs } \alpha\text{-conversion}}$.

α-conversion and β-reduction

Definition (α-conversion)

Let M be a λ-term, then

$$\lambda x.M = \lambda y.M[x \mapsto y]. \tag{1}$$

This is also called *alpha* equivalence.

α-conversion and β-reduction

Definition (α-conversion)

Let M be a λ-term, then

$$\lambda x.M = \lambda y.M[x \mapsto y]. \tag{1}$$

This is also called *alpha* equivalence.

Definition (β-reduction)

Let M and N be λ-terms, then

$$(\lambda x.M)\, N = M[x \mapsto N]. \tag{2}$$

We say that N is applied to $\lambda x.M$.

Free and Bounded Variables

Definition (Free Variables)

The set of *free variables* of a λ-term E, denoted $FV(E)$ is defined as

1. $FV(x) = x$;
2. $FV(M\ N) = FV(M) \cup FV(N)$;
3. $FV(\lambda x.M) = FV(M) \setminus x$.

Other variables are said to be *bounded*.

Definition (Combinator)

A λ-term E is said to be a *combinator* if, and only if, $FV(E) = \varnothing$.

Fixed Point Theorem (1)

Theorem (Fixed Point Theorem (1))

$$\forall_{F \in \Lambda} \exists_{X \in \Lambda} \quad F\ X = X \tag{3}$$

Fixed Point Theorem (1)

Theorem (Fixed Point Theorem (1))

$$\forall_{F \in \Lambda} \exists_{X \in \Lambda} \quad F\ X = X \tag{3}$$

Proof.

Let F be any abstraction in Λ. We prove that such X exists by letting $X = W\ W$, where $W = \lambda x.(F\ (x\ x))$. Then,

$$\begin{aligned} X &= W\ W && \text{(by def. of } X\text{)} \\ &= \lambda x.(F\ (x\ x))\ W && \text{(by def. of } W\text{)} \\ &= F\ (W\ W) && \text{(by } \beta\text{-reduction)} \\ &= F\ X. && \text{(by def. of X)} \end{aligned}$$

□

Fixed Point Theorem (2) — Y Combinator

Theorem (Fixed Point Theorem (2))

There is a fixed point combinator,

$$Y = \lambda f.(\lambda x.(f\ (x\ x))\ \lambda x.(f\ (x\ x))), \tag{4}$$

such that $\forall_{F \in \Lambda} \quad Y\ F = F\ (Y\ F)$.

Fixed Point Theorem (2) — Y Combinator

Theorem (Fixed Point Theorem (2))

There is a fixed point combinator,

$$Y = \lambda f.(\lambda x.(f\ (x\ x))\ \lambda x.(f\ (x\ x))), \tag{4}$$

such that $\forall_{F \in \Lambda} \quad Y\ F = F\ (Y\ F)$.

Proof.

$$\begin{aligned}
Y\ F &= \lambda f.(\lambda x.(f\ (x\ x))\ \lambda x.(f\ (x\ x)))\ F && \text{(by def. of } Y\text{)} \\
&= \lambda x.(F\ (x\ x))\ \lambda x.(F\ (x\ x)) && \text{(by } \beta\text{-reduction)} \\
&= F\ (\lambda x.(F\ (x\ x)\ \lambda x.(F\ (x\ x)) && \text{(by } \beta\text{-reduction)} \\
&= F\ (Y\ F) && \text{(by def. of } Y \text{ (and } \beta\text{-reduction))}
\end{aligned}$$

□

Practical Implications of the Fixed Point Theorem

The fixed point theorem, and in particular the Y combinator allows us to express recursive functions in *lambda*-calculus.

```
fact n = if n == 0 then 1 else n*(fact (n-1))
```

$$F = \underbrace{\lambda f.}_{\text{abstract from the recursion}} \lambda n.if\ (iszero\ n)\ 1\ (mul\ n\ (f\ (prec\ n)))$$

Then we can compute a factorial of a number k with $Y\ F\ k$.

Eg.

$$\begin{aligned} Y\ F\ 2 &= F\ (Y\ F)\ 2 \\ &= mul\ 2\ (Y\ F\ 1) \\ &= mul\ 2\ (F\ (Y\ F)\ 1) \\ &= mul\ 2\ (mul\ 1\ (Y\ F\ 0)) \\ &= mul\ 2\ (mul\ 1\ (F\ (Y\ F)\ 0)) \\ &= mul\ 2\ (mul\ 1\ 1) \end{aligned}$$

Church Numerals

Definition (Church Numerals)

We can encode natural numbers using in the following manner:

$$
\begin{aligned}
0 &\equiv \lambda f.\lambda x.x \\
1 &\equiv \lambda f.\lambda x.(f\ x) \\
2 &\equiv \lambda f.\lambda x.(f\ (f\ x)) \\
&\vdots \\
n &\equiv \lambda f.\lambda x.(f^n\ x)
\end{aligned} \tag{5}
$$

Encoding Booleans and Pairs

Definition (Booleans)

$$\text{true} \equiv \lambda x.\lambda y.x \tag{6}$$

$$\text{false} \equiv \lambda x.\lambda y.y \tag{7}$$

This definition easily expresses *if* conditionals.

Encoding Booleans and Pairs

Definition (Booleans)

$$\mathsf{true} \equiv \lambda x.\lambda y.x \tag{6}$$

$$\mathsf{false} \equiv \lambda x.\lambda y.y \tag{7}$$

This definition easily expresses *if* conditionals.

Definition (Pairs)

$$[M, N] \equiv \lambda z.z\ M\ N \tag{8}$$

By applying true or false we can extract the components of $[M, N]$: we denote $[M, N]_1 \equiv [M, N]\ \mathsf{true} = M$ and $[M, N]_2 \equiv [M, N]\ \mathsf{false} = N$.

Z, S, and P_i^n are λ-definable

Lemma (Base RPar functions are λ-definable)

The functions Z, S and P_i^n of RPar can be defined in λ-calculus.

Proof.

We give the following definition of Z, S and P_i^n:

$$Z \equiv 0 \equiv \lambda f.\lambda x.x \tag{9}$$

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

$$P_i^n \equiv \lambda x_1.\lambda x_2.\cdots\lambda x_n.x_i \tag{11}$$

□

Composition is λ-definable

Lemma (Composition is λ-definable)

The function $Comp(g, h_1, \cdots, h_n)(\vec{x}) = g(h_1(\vec{x}), \cdots, h_n(\vec{x}))$ *of RPar can be defined in* λ*-calculus.*

Proof.

Let $g, h_1, \cdots, h_n$ be lambda abstractions, then the composition function can be defined as,

$$Comp(g, h_1, \cdots, h_n) \equiv \lambda\vec{x}.(g\ (h_1\ \vec{x})\ \cdots\ (h_n\ \vec{x})) \tag{12}$$

□

RPrim is λ-definable

Lemma (*RPrim* is λ-definable)

The function $RPrim(g, h)(k, x) = f(k, x)$, *where*

$$f(0, \vec{x}) = g(\vec{x});$$
$$f(k+1, \vec{x}) = h(f(k, \vec{x}), k, \vec{x}),$$

can be defined in λ*-calculus.*

RPrim is λ-definable — Proof

Proof.

Let g and f be a lambda abstractions. We will define the function $F = f$. (For the sake of simplicity we omit the parameter $\vec{x}$.)

Let

$$T = \lambda p.[S\ p_1, h\ p_2\ p_1], \tag{13}$$

then

$$T\ [k, f'\ k] = [k+1, h\ (f'\ k)\ k]. \tag{14}$$

We define

$$F \equiv \lambda k.(k\ T\ [0, g])_2 \tag{15}$$

$$= (T^k\ [0, g\ 0])_2 \qquad \text{(by def. of Church Numerals)}$$

□

μ-recursion is λ-definable

Lemma (μ-recursion is λ-definable)

The μ operator is can be defined in λ-calculus.

Proof.

Let g be a lambda abstractions. Then we can define the μ operator as

$$\mu k[g(\vec{x}, k) = 0] \equiv (Y\ (\lambda f.\lambda k.if\ (iszero\ (g\ \vec{x}\ k))\ k\ (f\ (S\ k))))\ 0 \tag{16}$$

where

$$if = \lambda c.\lambda t.\lambda f.(c\ t\ f)$$
$$iszero = \lambda n.(n\ (\lambda x.\text{false})\ \text{true})$$

Lambda Calculus is Turing Complete

Theorem (λ-calculus Turing Completeness)

All recursive functions are λ-definable.

Proof.

By the previous lemmas every *RPar* expression can be represented by a λ-term. □

References (1)

Henk Barendregt.
Lambda Calculi with Types, volume 2.
1992.

F. Cardone and J. R. Hindley.
History of Lambda-Calculus and Combinatory Logic.
Handbook of the History of Logic, 5.